Lambda Student Resources Site

- 0-notes/
- 1-projects/
- 2-resources/
- 3-misc/
- 4-test-prep/
- 5-websites/
- 6-about/
- 7-assets/
- 8-site-documentation/
- 9-markdown/
- 10-miscellaneous/
- 12-Core-Site/
- 13-web-tools/
- css/
- docs/
- js/
- media-gifs-images/
- react-md/
- WEEKS

Lambda Student Resource Hub

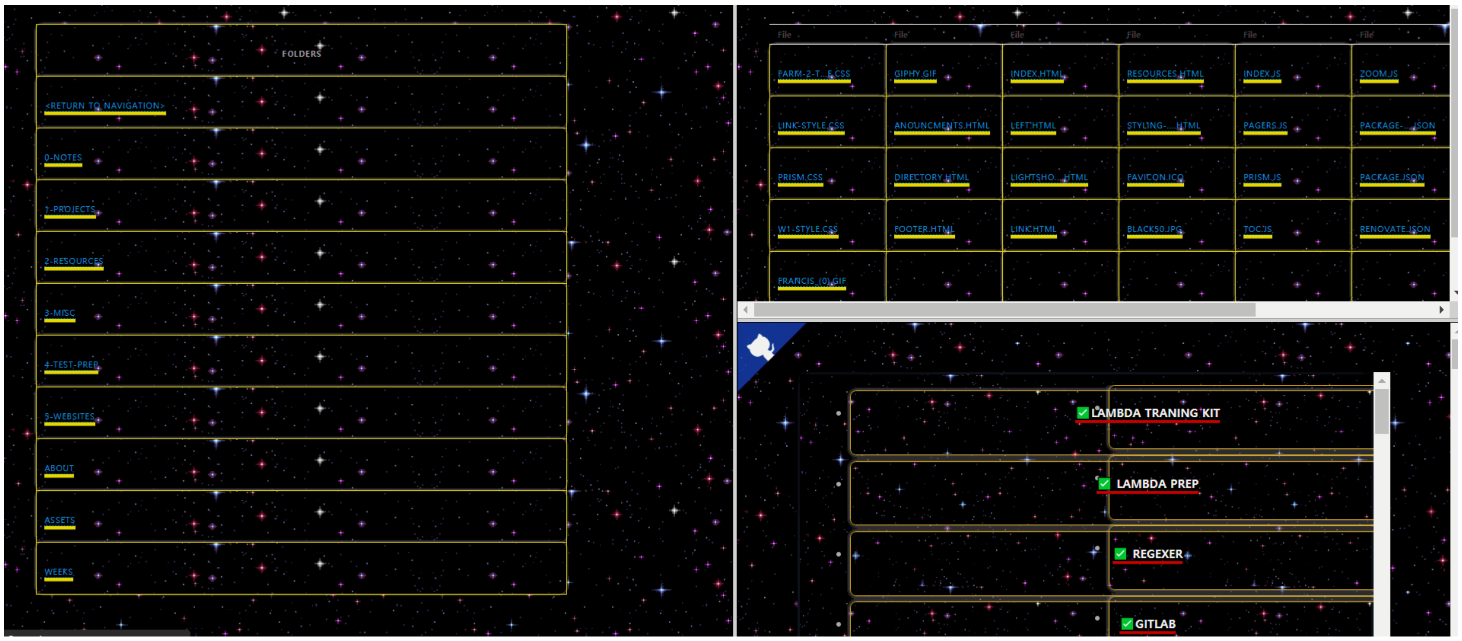
> Disclaimer: this repo is 2GB as of this writing and growing fast.... download at your own discretion.... if you want to download it but don't have the space the icloud storage section located at the top of the deployed website contains the entire contents of this repo as well as bonus materials!

Table of contents

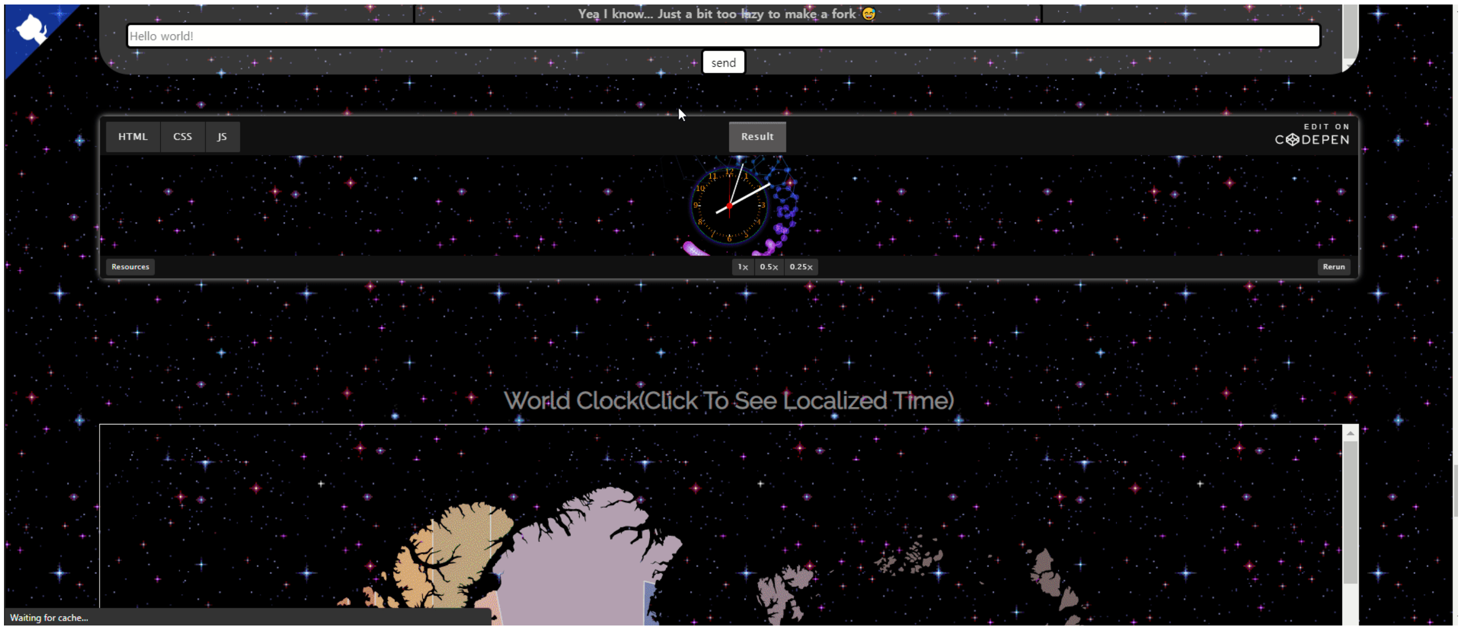
- Review-Of-Previous-Concepts
- Array-CB-Methods
- Mutability
- Objects
- Reiew
- Scope
- Understanding By Example
- Spread and Rest

-build-log

- Lambda Resources Website)

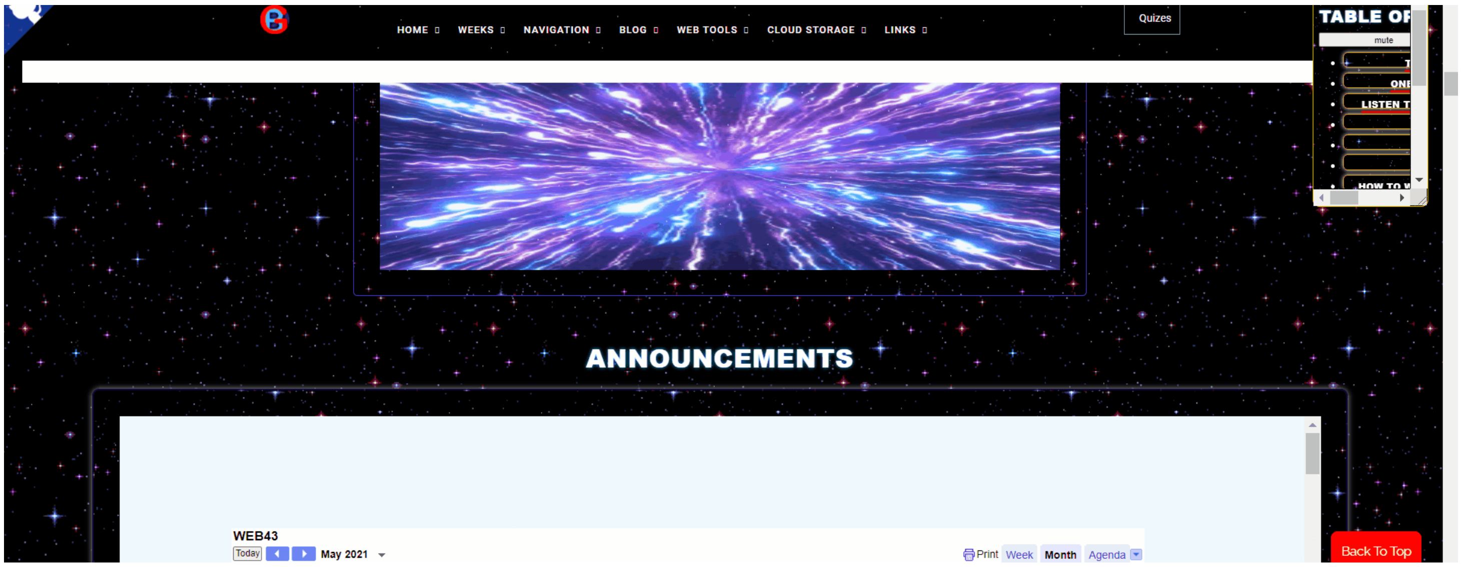


Lambda Resources Website



Loading Gif:

Embedable-Nav-Bar



WEB DEV RESOURCES:

Platforms)

- Node.js - Async non-blocking event-driven JavaScript runtime built on Chrome's V8 JavaScript engine.)
  - Cross-Platform - Writing cross-platform code on Node.js.)
- Frontend Development)
- iOS - Mobile operating system for Apple phones and tablets.)
- Android - Mobile operating system developed by Google.)
- IoT & Hybrid Apps)
- Electron - Cross-platform native desktop apps using JavaScript/HTML/CSS.)
- Cordova - JavaScript API for hybrid apps.)
- React Native - JavaScript framework for writing natively rendering mobile apps for iOS and Android.)
- Xamarin - Mobile app development IDE, testing, and distribution.)
- Linux)
  - Containers)
  - eBPF - Virtual machine that allows you to write more efficient and powerful tracing and monitoring for Linux systems.)
  - Arch-based Projects - Linux distributions and projects based on Arch Linux.)
- macOS - Operating system for Apple's Mac computers.)
  - Command-Line)
  - Screensavers)
  - Apps)
  - Open Source Apps)
- watchOS - Operating system for the Apple Watch.)
- JVM)
- Salesforce)
- Amazon Web Services)
- Windows)
- IPFS - P2P hypermedia protocol.)
- Fuse - Mobile development tools.)
- Heroku - Cloud platform as a service.)

- Raspberry Pi - Credit card-sized computer aimed at teaching kids programming, but capable of a lot more.)
- Qt - Cross-platform GUI app framework.)
- WebExtensions - Cross-browser extension system.)
- RubyMotion - Write cross-platform native apps for iOS, Android, macOS, tvOS, and watchOS in Ruby.)
- Smart TV - Create apps for different TV platforms.)
- GNOME - Simple and distraction-free desktop environment for Linux.)
- KDE - A free software community dedicated to creating an open and user-friendly computing experience.)
- . NET)
  - Core)
  - Roslyn - Open-source compilers and code analysis APIs for C# and VB. NET languages.)
- Amazon Alexa - Virtual home assistant.)
- DigitalOcean - Cloud computing platform designed for developers.)
- Flutter - Google's mobile SDK for building native iOS and Android apps from a single codebase written in Dart.)
- Home Assistant - Open source home automation that puts local control and privacy first.)
- IBM Cloud - Cloud platform for developers and companies.)
- Firebase - App development platform built on Google Cloud Platform.)
- Robot Operating System 2.0 - Set of software libraries and tools that help you build robot apps.)
- Adafruit IO - Visualize and store data from any device.)
- Cloudflare - CDN, DNS, DDoS protection, and security for your site.)
- Actions on Google - Developer platform for Google Assistant.)
- ESP - Low-cost microcontrollers with WiFi and broad IoT applications.)
- Deno - A secure runtime for JavaScript and TypeScript that uses V8 and is built in Rust.)
- DOS - Operating system for x86-based personal computers that was popular during the 1980s and early 1990s.)
- Nix - Package manager for Linux and other Unix systems that makes package management reliable and reproducible.)
)

Programming Languages)

)

- JavaScript)
  - Promises)
  - Standard Style - Style guide and linter.)

    - Must Watch Talks)
    - Tips)
    - Network Layer)
    - Micro npm Packages)
    - Mad Science npm Packages - Impossible sounding projects that exist.)
    - Maintenance Modules - For npm packages.)
    - npm - Package manager.)
    - AVA - Test runner.)
    - ESLint - Linter.)
    - Functional Programming)
    - Observables)
    - npm scripts - Task runner.)
    - 30 Seconds of Code - Code snippets you can understand in 30 seconds.)
    - Ponyfills - Like polyfills but without overriding native APIs.)
- Swift - Apple's compiled programming language that is secure, modern, programmer-friendly, and fast.)
    - Education)
    - Playgrounds)
- Python - General-purpose programming language designed for readability.)
    - Asyncio - Asynchronous I/O in Python 3.)
    - Scientific Audio - Scientific research in audio/music.)
    - CircuitPython - A version of Python for microcontrollers.)
    - Data Science - Data analysis and machine learning.)
    - Typing - Optional static typing for Python.)
    - MicroPython - A lean and efficient implementation of Python 3 for microcontrollers.)
- Rust)
- Haskell)
- PureScript)
- Go)
- Scala)
    - Scala Native - Optimizing ahead-of-time compiler for Scala based on LLVM.)
- Ruby)
- Clojure)
- ClojureScript)
- Elixir)
- Elm)
- Erlang)
- Julia - High-level dynamic programming language designed to address the needs of high-performance numerical analysis and computational science.)

- Lua)
- C)
- C/C++ - General-purpose language with a bias toward system programming and embedded, resource-constrained software.)
- R - Functional programming language and environment for statistical computing and graphics.)
    - Learning)
- D)
- Common Lisp - Powerful dynamic multiparadigm language that facilitates iterative and interactive development.)
    - Learning)
- Perl)
- Groovy)
- Dart)
- Java - Popular secure object-oriented language designed for flexibility to "write once, run anywhere".)
    - RxJava)
- Kotlin)
- OCaml)
- ColdFusion)
- Fortran)
- PHP - Server-side scripting language.)
    - Composer - Package manager.)
- Pascal)
- AutoHotkey)
- AutoIt)
- Crystal)
- Frege - Haskell for the JVM.)
- CMake - Build, test, and package software.)
- ActionScript 3 - Object-oriented language targeting Adobe AIR.)
- Eta - Functional programming language for the JVM.)
- Idris - General purpose pure functional programming language with dependent types influenced by Haskell and ML.)
- Ada/SPARK - Modern programming language designed for large, long-lived apps where reliability and efficiency are essential.)
- Q# - Domain-specific programming language used for expressing quantum algorithms.)
- Imba - Programming language inspired by Ruby and Python and compiles to performant JavaScript.)
- Vala - Programming language designed to take full advantage of the GLib and GNOME ecosystems, while preserving the speed of C code.)

- Coq - Formal language and environment for programming and specification which facilitates interactive development of machine-checked proofs.)
- V - Simple, fast, safe, compiled language for developing maintainable software.)
)

Front-End Development)

)

- ES6 Tools)
- Web Performance Optimization)
- Web Tools)
- CSS - Style sheet language that specifies how HTML elements are displayed on screen.)
  - Critical-Path Tools)
  - Scalability)
  - Must-Watch Talks)
  - Protips)
  - Frameworks)
- React - App framework.)
  - Relay - Framework for building data-driven React apps.)
  - React Hooks - A new feature that lets you use state and other React features without writing a class.)
- Web Components)
- Polymer - JavaScript library to develop Web Components.)
- Angular - App framework.)
- Backbone - App framework.)
- HTML5 - Markup language used for websites & web apps.)
- SVG - XML-based vector image format.)
- Canvas)
- KnockoutJS - JavaScript library.)
- Dojo Toolkit - JavaScript toolkit.)
- Inspiration)
- Ember - App framework.)
- Android UI)
- iOS UI)
- Meteor)
- BEM)
- Flexbox)
- Web Typography)

- Web Accessibility)
- Material Design)
- D3 - Library for producing dynamic, interactive data visualizations.)
- Emails)
- jQuery - Easy to use JavaScript library for DOM manipulation.)
    - Tips)
- Web Audio)
- Offline-First)
- Static Website Services)
- Cycle.js - Functional and reactive JavaScript framework.)
- Text Editing)
- Motion UI Design)
- Vue.js - App framework.)
- Marionette.js - App framework.)
- Aurelia - App framework.)
- Charting)
- Ionic Framework 2)
- Chrome DevTools)
- PostCSS - CSS tool.)
- Draft.js - Rich text editor framework for React.)
- Service Workers)
- Progressive Web Apps)
- choo - App framework.)
- Redux - State container for JavaScript apps.)
- webpack - Module bundler.)
- Browserify - Module bundler.)
- Sass - CSS preprocessor.)
- Ant Design - Enterprise-class UI design language.)
- Less - CSS preprocessor.)
- WebGL - JavaScript API for rendering 3D graphics.)
- Preact - App framework.)
- Progressive Enhancement)
- Next.js - Framework for server-rendered React apps.)
- lit-html - HTML templating library for JavaScript.)
- JAMstack - Modern web development architecture based on client-side JavaScript, reusable APIs, and prebuilt markup.)
- WordPress-Gatsby - Web development technology stack with WordPress as a back end and Gatsby as a front end.)
- Mobile Web Development - Creating a great mobile web experience.)

- Storybook - Development environment for UI components.)
- Blazor - . NET web framework using C#/Razor and HTML that runs in the browser with WebAssembly.)
- PageSpeed Metrics - Metrics to help understand page speed and user experience.)
- Tailwind CSS - Utility-first CSS framework for rapid UI development.)
- Seed - Rust framework for creating web apps running in WebAssembly.)
- Web Performance Budget - Techniques to ensure certain performance metrics for a website.)
- Web Animation - Animations in the browser with JavaScript, CSS, SVG, etc.)
- Yew - Rust framework inspired by Elm and React for creating multi-threaded frontend web apps with WebAssembly.)
- Material-UI - Material Design React components for faster and easier web development.)
- Building Blocks for Web Apps - Standalone features to be integrated into web apps.)
- Svelte - App framework.)
- Design systems - Collection of reusable components, guided by rules that ensure consistency and speed.)
)

Back-End Development)

)

- Flask - Python framework.)
- Docker)
- Vagrant - Automation virtual machine environment.)
- Pyramid - Python framework.)
- Play1 Framework)
- CakePHP - PHP framework.)
- Symfony - PHP framework.)
  - Education)
- Laravel - PHP framework.)
  - Education)
  - TALL Stack - Full-stack development solution featuring libraries built by the Laravel community.)
- Rails - Web app framework for Ruby.)
  - Gems - Packages.)
- Phalcon - PHP framework.)
- Useful `.htaccess` Snippets)
- nginx - Web server.)
- Dropwizard - Java framework.)

- Kubernetes - Open-source platform that automates Linux container operations.)
- Lumen - PHP micro-framework.)
- Serverless Framework - Serverless computing and serverless architectures.)
- Apache Wicket - Java web app framework.)
- Vert.x - Toolkit for building reactive apps on the JVM.)
- Terraform - Tool for building, changing, and versioning infrastructure.)
- Vapor - Server-side development in Swift.)
- Dash - Python web app framework.)
- FastAPI - Python web app framework.)
- CDK - Open-source software development framework for defining cloud infrastructure in code.)
- IAM - User accounts, authentication and authorization.)
)

Computer Science)

)

- University Courses)
- Data Science)
  - Tutorials)
- Machine Learning)
  - Tutorials)
  - ML with Ruby - Learning, implementing, and applying Machine Learning using Ruby.)
  - Core ML Models - Models for Apple's machine learning framework.)
  - H2O - Open source distributed machine learning platform written in Java with APIs in R, Python, and Scala.)
  - Software Engineering for Machine Learning - From experiment to production-level machine learning.)
  - AI in Finance - Solving problems in finance with machine learning.)
  - JAX - Automatic differentiation and XLA compilation brought together for high-performance machine learning research.)
- Speech and Natural Language Processing)
  - Spanish)
  - NLP with Ruby)
  - Question Answering - The science of asking and answering in natural language with a machine.)
  - Natural Language Generation - Generation of text used in data to text, conversational agents, and narrative generation applications.)
- Linguistics)

- Cryptography)
  - Papers - Theory basics for using cryptography by non-cryptographers.)
- Computer Vision)
- Deep Learning - Neural networks.)
  - TensorFlow - Library for machine intelligence.)
  - TensorFlow.js - WebGL-accelerated machine learning JavaScript library for training and deploying models.)
  - TensorFlow Lite - Framework that optimizes TensorFlow models for on-device machine learning.)
  - Papers - The most cited deep learning papers.)
  - Education)
- Deep Vision)
- Open Source Society University)
- Functional Programming)
- Empirical Software Engineering - Evidence-based research on software systems.)
- Static Analysis & Code Quality)
- Information Retrieval - Learn to develop your own search engine.)
- Quantum Computing - Computing which utilizes quantum mechanics and qubits on quantum computers.)
)

Big Data)

)

- Big Data)
- Public Datasets)
- Hadoop - Framework for distributed storage and processing of very large data sets.)
- Data Engineering)
- Streaming)
- Apache Spark - Unified engine for large-scale data processing.)
- Qlik - Business intelligence platform for data visualization, analytics, and reporting apps.)
- Splunk - Platform for searching, monitoring, and analyzing structured and unstructured machine-generated big data in real-time.)
)

Theory)

)

- Papers We Love)
- Talks)
- Algorithms)
  - Education - Learning and practicing.)
- Algorithm Visualizations)
- Artificial Intelligence)
- Search Engine Optimization)
- Competitive Programming)
- Math)
- Recursion Schemes - Traversing nested data structures.)
)

Books)

)

- Free Programming Books)
- Go Books)
- R Books)
- Mind Expanding Books)
- Book Authoring)
- Elixir Books)
)

Editors)

)

- Sublime Text)
- Vim)
- Emacs)
- Atom - Open-source and hackable text editor.)
- Visual Studio Code - Cross-platform open-source text editor.)
)

Gaming)

)

- Game Development)
- Game Talks)
- Godot - Game engine.)
- Open Source Games)
- Unity - Game engine.)
- Chess)
- LÖVE - Game engine.)
- PICO-8 - Fantasy console.)
- Game Boy Development)
- Construct 2 - Game engine.)
- Gideros - Game engine.)
- Minecraft - Sandbox video game.)
- Game Datasets - Materials and datasets for Artificial Intelligence in games.)
- Haxe Game Development - A high-level strongly typed programming language used to produce cross-platform native code.)
- libGDX - Java game framework.)
- PlayCanvas - Game engine.)
- Game Remakes - Actively maintained open-source game remakes.)
- Flame - Game engine for Flutter.)
- Discord Communities - Chat with friends and communities.)
- CHIP-8 - Virtual computer game machine from the 70s.)
- Games of Coding - Learn a programming language by making games.)
)

Development Environment)

)

- Quick Look Plugins - For macOS.)
- Dev Env)
- Dotfiles)
- Shell)
- Fish - User-friendly shell.)
- Command-Line Apps)
- ZSH Plugins)
- GitHub - Hosting service for Git repositories.)
  - Browser Extensions)

    - Cheat Sheet)
    - Pinned Gists - Dynamic pinned gists for your GitHub profile.)
- Git Cheat Sheet & Git Flow)
- Git Tips)
- Git Add-ons - Enhance the `git` CLI.)
- Git Hooks - Scripts for automating tasks during `git` workflows.)
- SSH)
- FOSS for Developers)
- Hyper - Cross-platform terminal app built on web technologies.)
- PowerShell - Cross-platform object-oriented shell.)
- Alfred Workflows - Productivity app for macOS.)
- Terminals Are Sexy)
- GitHub Actions - Create tasks to automate your workflow and share them with others on GitHub.)
)

Entertainment)

)

- Science Fiction - Scifi.)
- Fantasy)
- Podcasts)
- Email Newsletters)
- IT Quotes)
)

Databases)

)

- Database)
- MySQL)
- SQLAlchemy)
- InfluxDB)
- Neo4j)
- MongoDB - NoSQL database.)
- RethinkDB)
- TinkerPop - Graph computing framework.)
- PostgreSQL - Object-relational database.)

- CouchDB - Document-oriented NoSQL database.)
- HBase - Distributed, scalable, big data store.)
- NoSQL Guides - Help on using non-relational, distributed, open-source, and horizontally scalable databases.)
- Contexture - Abstracts queries/filters and results/aggregations from different backing data stores like ElasticSearch and MongoDB.)
- Database Tools - Everything that makes working with databases easier.)
- Grakn - Logical database to organize large and complex networks of data as one body of knowledge.)
)

Media)

)

- Creative Commons Media)
- Fonts)
- Codeface - Text editor fonts.)
- Stock Resources)
- GIF - Image format known for animated images.)
- Music)
- Open Source Documents)
- Audio Visualization)
- Broadcasting)
- Pixel Art - Pixel-level digital art.)
- FFmpeg - Cross-platform solution to record, convert and stream audio and video.)
- Icons - Downloadable SVG/PNG/font icon projects.)
- Audiovisual - Lighting, audio and video in professional environments.)
)

Learn)

)

- CLI Workshoppers - Interactive tutorials.)
- Learn to Program)
- Speaking)
- Tech Videos)
- Dive into Machine Learning)

- Computer History)
- Programming for Kids)
- Educational Games - Learn while playing.)
- JavaScript Learning)
- CSS Learning - Mainly about CSS - the language and the modules.)
- Product Management - Learn how to be a better product manager.)
- Roadmaps - Gives you a clear route to improve your knowledge and skills.)
- YouTubers - Watch video tutorials from YouTubers that teach you about technology.)
)

Security)

)

- Application Security)
- Security)
- CTF - Capture The Flag.)
- Malware Analysis)
- Android Security)
- Hacking)
- Honeypots - Deception trap, designed to entice an attacker into attempting to compromise the information systems in an organization.)
- Incident Response)
- Vehicle Security and Car Hacking)
- Web Security - Security of web apps & services.)
- Lockpicking - The art of unlocking a lock by manipulating its components without the key.)
- Cybersecurity Blue Team - Groups of individuals who identify security flaws in information technology systems.)
- Fuzzing - Automated software testing technique that involves feeding pseudo-randomly generated input data.)
- Embedded and IoT Security)
- GDPR - Regulation on data protection and privacy for all individuals within EU.)
- DevSecOps - Integration of security practices into DevOps.)
)

Content Management Systems)

)

- Umbraco)
- Refinery CMS - Ruby on Rails CMS.)
- Wagtail - Django CMS focused on flexibility and user experience.)
- Textpattern - Lightweight PHP-based CMS.)
- Drupal - Extensible PHP-based CMS.)
- Craft CMS - Content-first CMS.)
- Sitecore - . NET digital marketing platform that combines CMS with tools for managing multiple websites.)
- Silverstripe CMS - PHP MVC framework that serves as a classic or headless CMS.)
)

Hardware)

)

- Robotics)
- Internet of Things)
- Electronics - For electronic engineers and hobbyists.)
- Bluetooth Beacons)
- Electric Guitar Specifications - Checklist for building your own electric guitar.)
- Plotters - Computer-controlled drawing machines and other visual art robots.)
- Robotic Tooling - Free and open tools for professional robotic development.)
- LIDAR - Sensor for measuring distances by illuminating the target with laser light.)
)

Business)

)

- Open Companies)
- Places to Post Your Startup)
- OKR Methodology - Goal setting & communication best practices.)
- Leading and Managing - Leading people and being a manager in a technology company/environment.)
- Indie - Independent developer businesses.)
- Tools of the Trade - Tools used by companies on Hacker News.)
- Clean Tech - Fighting climate change with technology.)
- Wardley Maps - Provides high situational awareness to help improve strategic planning and decision making.)

- Social Enterprise - Building an organization primarily focused on social impact that is at least partially self-funded.)
- Engineering Team Management - How to transition from software development to engineering management.)
- Developer-First Products - Products that target developers as the user.)
)

Work)

)

- Slack - Team collaboration.)
  - Communities)
- Remote Jobs)
- Productivity)
- Niche Job Boards)
- Programming Interviews)
- Code Review - Reviewing code.)
- Creative Technology - Businesses & groups that specialize in combining computing, design, art, and user experience.)
)

Networking)

)

- Software-Defined Networking)
- Network Analysis)
- PCAPTools)
- Real-Time Communications - Network protocols for near simultaneous exchange of media and data.)
)

Decentralized Systems)

)

- Bitcoin - Bitcoin services and tools for software developers.)
- Ripple - Open source distributed settlement network.)

- Non-Financial Blockchain - Non-financial blockchain applications.)
- Mastodon - Open source decentralized microblogging network.)
- Ethereum - Distributed computing platform for smart contract development.)
- Blockchain AI - Blockchain projects for artificial intelligence and machine learning.)
- EOSIO - A decentralized operating system supporting industrial-scale apps.)
- Corda - Open source blockchain platform designed for business.)
- Waves - Open source blockchain platform and development toolset for Web 3.0 apps and decentralized solutions.)
- Substrate - Framework for writing scalable, upgradeable blockchains in Rust.)
)

Higher Education)

)

- Computational Neuroscience - A multidisciplinary science which uses computational approaches to study the nervous system.)
- Digital History - Computer-aided scientific investigation of history.)
- Scientific Writing - Distraction-free scientific writing with Markdown, reStructuredText and Jupyter notebooks.)
)

Events)

)

- Creative Tech Events - Events around the globe for creative coding, tech, design, music, arts and cool stuff.)
- Events in Italy - Tech-related events in Italy.)
- Events in the Netherlands - Tech-related events in the Netherlands.)
)

Testing)

)

- Testing - Software testing.)
- Visual Regression Testing - Ensures changes did not break the functionality or style.)
- Selenium - Open-source browser automation framework and ecosystem.)

- Appium - Test automation tool for apps.)
- TAP - Test Anything Protocol.)
- JMeter - Load testing and performance measurement tool.)
- k6 - Open-source, developer-centric performance monitoring and load testing solution.)
- Playwright - Node.js library to automate Chromium, Firefox and WebKit with a single API.)
- Quality Assurance Roadmap - How to start & build a career in software testing.)
)

Miscellaneous)

)

- JSON - Text based data interchange format.)
  - GeoJSON)
  - Datasets)
- CSV - A text file format that stores tabular data and uses a comma to separate values.)
- Discounts for Student Developers)
- Radio)
- Awesome - Recursion illustrated.)
- Analytics)
- REST)
- Continuous Integration and Continuous Delivery)
- Services Engineering)
- Free for Developers)
- Answers - Stack Overflow, Quora, etc.)
- Sketch - Design app for macOS.)
- Boilerplate Projects)
- Readme)
- Design and Development Guides)
- Software Engineering Blogs)
- Self Hosted)
- FOSS Production Apps)
- Gulp - Task runner.)
- AMA - Ask Me Anything.)
  - Answers)
- Open Source Photography)
- OpenGL - Cross-platform API for rendering 2D and 3D graphics.)
- GraphQL)
- Transit)

- Research Tools)
- Data Visualization)
- Social Media Share Links)
- Microservices)
- Unicode - Unicode standards, quirks, packages and resources.)
  - Code Points)
- Beginner-Friendly Projects)
- Katas)
- Tools for Activism)
- Citizen Science - For community-based and non-institutional scientists.)
- MQTT - "Internet of Things" connectivity protocol.)
- Hacking Spots)
- For Girls)
- Vorpal - Node.js CLI framework.)
- Vulkan - Low-overhead, cross-platform 3D graphics and compute API.)
- LaTeX - Typesetting language.)
- Economics - An economist's starter kit.)
- Funny Markov Chains)
- Bioinformatics)
- Cheminformatics - Informatics techniques applied to problems in chemistry.)
- Colorful - Choose your next color scheme.)
- Steam - Digital distribution platform.)
- Bots - Building bots.)
- Site Reliability Engineering)
- Empathy in Engineering - Building and promoting more compassionate engineering cultures.)
- DTrace - Dynamic tracing framework.)
- Userscripts - Enhance your browsing experience.)
- Pokémon - Pokémon and Pokémon GO.)
- ChatOps - Managing technical and business operations through a chat.)
- Falsehood - Falsehoods programmers believe in.)
- Domain-Driven Design - Software development approach for complex needs by connecting the implementation to an evolving model.)
- Quantified Self - Self-tracking through technology.)
- SaltStack - Python-based config management system.)
- Web Design - For digital designers.)
- Creative Coding - Programming something expressive instead of something functional.)
- No-Login Web Apps - Web apps that work without login.)
- Free Software - Free as in freedom.)
- Framer - Prototyping interactive UI designs.)

- Markdown - Markup language.)
- Dev Fun - Funny developer projects.)
- Healthcare - Open source healthcare software for facilities, providers, developers, policy experts, and researchers.)
- Magento 2 - Open Source eCommerce built with PHP.)
- TikZ - Graph drawing packages for TeX/LaTeX/ConTeXt.)
- Neuroscience - Study of the nervous system and brain.)
- Ad-Free - Ad-free alternatives.)
- Esolangs - Programming languages designed for experimentation or as jokes rather than actual use.)
- Prometheus - Open-source monitoring system.)
- Homematic - Smart home devices.)
- Ledger - Double-entry accounting on the command-line.)
- Web Monetization - A free open web standard service that allows you to send money directly in your browser.)
- Uncopyright - Public domain works.)
- Crypto Currency Tools & Algorithms - Digital currency where encryption is used to regulate the generation of units and verify transfers.)
- Diversity - Creating a more inclusive and diverse tech community.)
- Open Source Supporters - Companies that offer their tools and services for free to open source projects.)
- Design Principles - Create better and more consistent designs and experiences.)
- Theravada - Teachings from the Theravada Buddhist tradition.)
- inspectIT - Open source Java app performance management tool.)
- Open Source Maintainers - The experience of being an open source maintainer.)
- Calculators - Calculators for every platform.)
- Captcha - A type of challenge-response test used in computing to determine whether or not the user is human.)
- Jupyter - Create and share documents that contain code, equations, visualizations and narrative text.)
- FIRST Robotics Competition - International high school robotics championship.)
- Humane Technology - Open source projects that help improve society.)
- Speakers - Conference and meetup speakers in the programming and design community.)
- Board Games - Table-top gaming fun for all.)
- Software Patreons - Fund individual programmers or the development of open source projects.)
- Parasite - Parasites and host-pathogen interactions.)
- Food - Food-related projects on GitHub.)
- Mental Health - Mental health awareness and self-care in the software industry.)
- Bitcoin Payment Processors - Start accepting Bitcoin.)

- Scientific Computing - Solving complex scientific problems using computers.)
- Amazon Sellers)
- Agriculture - Open source technology for farming and gardening.)
- Product Design - Design a product from the initial concept to production.)
- Prisma - Turn your database into a GraphQL API.)
- Software Architecture - The discipline of designing and building software.)
- Connectivity Data and Reports - Better understand who has access to telecommunication and internet infrastructure and on what terms.)
- Stacks - Tech stacks for building different apps and features.)
- Cytodata - Image-based profiling of biological phenotypes for computational biologists.)
- IRC - Open source messaging protocol.)
- Advertising - Advertising and programmatic media for websites.)
- Earth - Find ways to resolve the climate crisis.)
- Naming - Naming things in computer science done right.)
- Biomedical Information Extraction - How to extract information from unstructured biomedical data and text.)
- Web Archiving - An effort to preserve the Web for future generations.)
- WP-CLI - Command-line interface for WordPress.)
- Credit Modeling - Methods for classifying credit applicants into risk classes.)
- Ansible - A Python-based, open source IT configuration management and automation platform.)
- Biological Visualizations - Interactive visualization of biological data on the web.)
- QR Code - A type of matrix barcode that can be used to store and share a small amount of information.)
- Veganism - Making the plant-based lifestyle easy and accessible.)
- Translations - The transfer of the meaning of a text from one language to another.)
)

Related)

)

- All Awesome Lists - All the Awesome lists on GitHub.)
- Awesome Indexed - Search the Awesome dataset.)
- Awesome Search - Quick search for Awesome lists.)
- StumbleUponAwesome - Discover random pages from the Awesome dataset using a browser extension.)
- Awesome CLI - A simple command-line tool to dive into Awesome lists.)
- Awesome Viewer - A visualizer for all of the above Awesome lists.)

Running Resource List:

> The outward or visible aspect of a website.

The outward or visible aspect of a website.

- **Animation**: The process of creating motion and shape change.
  - **Animate.css**: Just-add-water CSS animations.
  - **Animate.less**: A bunch of cool, fun, and cross-browser animations converted into LESS for you to use in your Bootstrap projects.
  - **Anime.js**: Anime is a flexible yet lightweight JavaScript animation library. It works with CSS, Individual Transforms, SVG, DOM attributes and JS Objects.
  - **Approach**: A jQuery plugin that allows you to animate CSS properties based on distance to an object.
  - **CSS Spritesheet Animation Example**: Sprite Sheet animation with CSS3 using the steps() feature.
  - **Caat**: Scene graph director-based animation framework for javascript.
  - **CanvasScript3**: CanvasScript3 is a Javascript library for the new HTML5 Canvas with an interface similar to ActionScript3. This library enables Sprite Groups, Layers, Mouse Events, Keyboard Events, Bitmap Effects, Tween Animations etc.
  - **Collie**: Collie is a Javascript library that helps to create highly optimized animations and games using HTML 5. Collie runs on both PC and mobile using HTML 5 canvas and DOM.
  - **Emile.js**: Emile.js is a no-frills stand-alone CSS animation JavaScript framework.
  - **Firmin**: Firmin is a JavaScript animation library using CSS transforms and transitions.
  - **GreenSock Animation Platform**: GreenSock Animation Platform is a suite of tools for scripted animation.
    - **Codepen Repository**: Codepen repository with examples of Greensock usage and code.
    - **Examples**: Here are a couple of examples demonstrating the core features of the Greensock Animation Platform.
    - **Learning Center**: Tutorials and videos for GreenSock Animation Platform.
  - **JQuery Transit**: Super-smooth CSS3 transformations and transitions for jQuery.

    - **Janis**: Janis is a lightweight Javascript framework that provides simple animations via CSS transitions for modern browsers on the web as well as mobile devices.
    - **Keanu**: Keanu is a micro-lib for animation on Canvas/JS.
    - **Magic**: CSS3 Animations with special effects.
    - **Move.js**: Move.js is a small JavaScript library making CSS3 backed animation extremely simple and elegant.
    - **Ramjet**: Ramjet makes it looks as though one DOM element is capable of transforming into another, no matter where the two elements sit in the DOM tree.
    - **Rekapi**: A keyframe animation library for JavaScript.
    - **SVG.js**: A lightweight library for manipulating and animating SVG.
    - **Scripty2**: scripty2 is a powerful, flexible JavaScript framework to help you write your own delicious visual effects & user interfaces.
    - **Shifty**: Shifty is a tweening engine built in JavaScript. It is designed to fit any number of tweening needs.
    - **Snap.svg**: Snap.svg JavaScript library makes working with your SVG assets as easy as jQuery makes working with the DOM.
    - **Stylie**: Stylie is a fun tool for easily creating complex CSS animations. Quickly design your animation graphically, grab the generated code and go!
    - **Textillate.js**: Textillate.js combines some awesome libraries to provide a ease-to-use plugin for applying CSS3 animations to any text.
    - **Tween.js**: Super simple, fast and easy to use tweening engine which incorporates optimised Robert Penner's equations.
    - **Twitter Fave Animation**: Rather than rely on CSS transitions, the new animation makes use of a series of images. Here’s how to recreate the animation using the CSS animation steps timing function.
    - **Web Animation Past, Present, and Future (2016)**: Rachel Nabors explores the world of web animation standards, platforms and tools in 2016: SVG, SMIL, GreenSock AP, Framer, Browser Tooling etc.
    - **Web Animations API**: Web Animations is a new JavaScript API for driving animated content on the web. By unifying the animation features of SVG and CSS, Web Animations unlocks features previously only usable declaratively, and exposes powerful, high-performance animation capabilities to developers.
        - **Are we animated yet?**: This page tracks the progress of implementing the Web Animations API in Firefox.
        - **WAAPI Browser Support Test (+ Polyfill)**: This codepen tests whether and to which extend your browser supports Web Animations API. The test is run after including the Polyfill.
        - **Web Animations Polyfill**: JavaScript implementation of the Web Animations API.
- **Typography**: The style, arrangement, or appearance of typeset matter.

- **A Comprehensive Guide to Font Loading Strategies**: Zach Leatherman describes different approaches to loading of web fonts.
- **Adobe Edge Web Fonts**: Edge Web Fonts is a free service that provides access to a large library of fonts for your website. It’s one of the Edge Tools & Services from Adobe. Use of the service is free and unlimited.
- **Baseline.js**: A simple jQuery plugin for restoring vertical baselines thrown off by odd image sizes.
- **CSS Typography cheat sheet**: Small roundup on CSS features that will enhance your web typography.
- **Convincing a browser to load fonts from other domains**: A StackOverflow question about loading fonts across domains.
- **FitText**: FitText makes font-sizes flexible. Use this plugin on your fluid or responsive layout to achieve scalable headlines that fill the width of a parent element.
- **FlowType. JS**: Font-size and line-height based on element width.
- **Fontmatrix**: Matrix of fonts bundled with Mac and Windows operating systems, Microsoft Office and Adobe Creative Suite.
- **Google Fonts**: Google Fonts makes it quick and easy for everyone to use web fonts. Our goal is to create a directory of web fonts for the world to use. Our API service makes it easy to add Google Fonts to a website in seconds.
- **Gutenberg**: Gutenberg is a flexible and simple-to-use web typography starter kit for web designers and developers.
- **Lettering.js**: Web type is exploding all over the web but CSS currently doesn't offer complete down-to-the-letter control. So we created a jQuery plugin to give you that control.
- **OpenFoundry**: A platform for open-source fonts in a noise-free environment; to highlight their beauty and encourage further exploration.
- **Pure Typography**: CSS Styles for nicer web type. Depends on Pure.
- **Quick guide to webfonts via @font-face**: The @font-face feature from CSS3 allows us to use custom typefaces on the web in an accessible, manipulable, and scalable way.
- **Truly Fluid Typography With vh And vw Units**: This article describes viewport units and other technics to achieve typography which resizes smoothly with the screen.
- **TypeButter**: TypeButter allows you to set optical kerning for any font on your website. If you're longing for beautifully laid out text that today' browsers just don't provide, this is the plugin for you.
- **Typeset.css**: A no-nonsense CSS typography reset for styling user-generated content like blog posts, comments, and forum content.
- **Typeset.css**: A Sass library that provides some sensible default styles, optional classes to use & extend as needed, and some utility functions & mixins to make elevating your typography simpler.
- **bacon**: Bacon is a jQuery plugin that allows you to wrap text around a bezier curve or a line.

    - **slabText**: A jQuery plugin for producing big, bold & responsive headlines.
    - **trunk8**: trunk8 is an intelligent text truncation plugin to jQuery. When applied to a large block of text, trunk8 will cut off just enough text to prevent it from spilling over.
- **Visualization**: Placing data in a visual context.
    - **Bonsai.js**: A lightweight graphics library with an intuitive graphics API and an SVG renderer.
    - **Chart.js**: Simple, clean and engaging charts for designers and developers.
    - **Crossfilter**: Crossfilter is a JavaScript library for exploring large multivariate datasets in the browser.
    - **Cube**: Cube is a system for collecting timestamped events and deriving metrics. By collecting events rather than metrics, Cube lets you compute aggregate statistics post hoc.
    - **Cubism.js**: Cubism.js is a D3 plugin for visualizing time series. Use Cubism to construct better realtime dashboards, pulling data from Graphite, Cube and other sources.
    - **D3.js**: D3.js is a JavaScript library for manipulating documents based on data. D3 helps you bring data to life using HTML, SVG, and CSS.
        - **DataMaps**: Customizable SVG (world) map visualizations for the web in a single Javascript file using D3.js.
        - **Interactive Introduction to D3**: D3 slides in D3 that I put together after becoming frustrated with explaining D3 using PowerPoint.
        - **NVD3**: This project is an attempt to build re-usable charts and chart components for d3.js without taking away the power that d3.js gives you.
        - **Tutorial: Introduction to D3**: Basically we just plot hidden circles randomly on the screen, and then transition them to a portion of the screen. Then we add some interaction to it so that the circles will move once you move your mouse over them.
        - **xCharts**: xCharts is a JavaScript library for building beautiful and custom data-driven chart visualizations for the web using D3.js. Using HTML, CSS, and SVG, xCharts are designed to be dynamic, fluid, and open to integrations and customization.
    - **Easy Pie Chart**: Easy pie chart is a jQuery plugin that uses the canvas element to render simple pie charts for single values
    - **Flot**: Flot is a pure JavaScript plotting library for jQuery, with a focus on simple usage, attractive looks and interactive features.
    - **Google Chart Tools**: The Google Visualization API allows you to create charts and reporting applications over structured data and helps integrate these directly into your website.
    - **Paper.js**: Paper.js offers a clean Scene Graph / Document Object Model and a lot of powerful functionality to create and work with vector graphics and bezier curves.
    - **Photon**: Photon is a JavaScript library that adds simple lighting effects to DOM elements in 3D space.
    - **Piecon**: A tiny javascript library for dynamically generating progress pie charts in your favicons.

    - **Processing.js**: Processing.js is the sister project of the popular Processing visual programming language, designed for the web. Processing.js makes your data visualizations work using web standards.
    - **Smoothie Charts**: A JavaScript Charting Library for Streaming Data.
    - **TimelineJS**: TimelineJS is an open-source tool that enables anyone to build visually rich, interactive timelines.
    - **Timesheet.js**: Visualize your data and events with sexy HTML5 and CSS3. Create simple time sheets with sneaky JavaScript. Style them with CSS and have mobile fun as well.
    - **Treefun by Jim Blackler**: This tool creates SVG (Standard Vector Graphics) files to illustrate information structured as a basic tree.
    - **jQuery. Gantt**: Draw Gantt charts with the famous jQuery ease of development.
    - **jStat**: jStat is a statistical library written in JavaScript that allows you to perform advanced statistical operations without the need of a dedicated statistical language (e.g. MATLAB or R).
    - **morris.js**: Morris.js is a very simple API for drawing line, bar, area and donut charts.
    - **mxgraph**: mxGraph is a JavaScript diagramming library that enables interactive graph and charting applications to be quickly created that run natively in any major browser, both HTML 5 capable and Internet Explorer v7+.
    - **three.js**: Three.js is a library that makes WebGL - 3D in the browser - easy to use. While a simple cube in raw WebGL would turn out hundreds of lines of Javascript and shader code, a Three.js equivalent is only a fraction of that.
    - **vis.js**: Vis.js is a dynamic, browser based visualization library. The library is designed to be easy to use, handle large amounts of dynamic data, and enable manipulation of the data.

High level structure of the frontend code and the discipline of creating such structures.

- **Algorithms**: A self-contained step-by-step set of operations to be performed. Algorithms perform calculation, data processing, and/or automated reasoning tasks.
    - **Algorithm Visualizer**: A collection of algorithms with code and visualizations for each one of them.
    - **Sorting Algorithms Animations**: The following animations illustrate how effectively data sets from different starting points can be sorted using different algorithms.
- **Design Patterns**: Best practices that the programmer can use to solve common problems when designing an application or system.
    - **CSS Modules**: A CSS Module is a CSS file in which all class names and animation names are scoped locally by default.
        - **CSS Modules Documentation**: General overview and some implementations.
        - **ES CSS Modules**: PostCSS plugin that combines CSS Modules and ES Imports.
        - **Tree Shaking Bootstrap**: Jacob Parker describes how to include only those parts of Bootstrap you are really using on your website by leveraging CSS modules and ES6

modules.

- **Components**: Reusable and composable pieces of HTML, CSS and/or JavaScript code which are mostly used for GUI elements.
  - **Component Check**: In this project Donald Pipowitch compares the usage and development of components in several frameworks such as Angular, Ember, Cycle.js and React.
  - **Container Components**: Container Components is a pattern which allows to separate data-fetching and rendering concerns and increase the reusability of the (child) components.
  - **Devbridge Styleguide**: Devbridge Styleguide helps you create, share, and automate a living visual style library of your brand.
  - **Presentational and Container Components**: Dan Abramov creates a pattern for separating presentational and container components to increase reusability and clarity of the application code.
  - **Web Components**: Web Components is a W3C standard for encapsulated, reusable and composable widgets for the web platform.
    - **Are We Componentized Yet?**: Tracking the progress of Web Components through standardisation, polyfillification and implementation.
    - **Custom Elements**: Eric Bidelman describes how to create new HTML elements and manage their life cycle.
    - **Custom Elements W3C Editor's Draft**: This specification describes the method for enabling the author to define and use new types of DOM elements in a document.
    - **HTML Imports W3C Editor's Draft**: HTML Imports are a way to include and reuse HTML documents in other HTML documents.
    - **HTML Imports: #include for the web**: Eric Bidelman describes how to use HTML imports and goes through several edge cases.
    - **HTML's New Template Tag**: The template element allows you to declare fragments of DOM which are parsed, inert at page load, and instantiated later at runtime.
    - **Shadow DOM 101**: Dominic Cooney shows you how to use Shadow DOM in this tutorial.
    - **Shadow DOM 201**: Eric Bidelman explains advanced topics related to styling of Shadow DOM elements.
    - **Shadow DOM 301**: Eric Bidelman talks about advanced Shadow DOM topics like multiple shadow roots, insertion points, event model and Shadow DOM Visualizer.
    - **Shadow DOM W3C Editor's Draft**: This specification describes a method of combining multiple DOM trees into one hierarchy and how these trees interact with each other within a document, thus enabling better composition of the DOM.
    - **ShadowDOM Visualizer**: This tool allows you to visualize how Shadow DOM renders in the browser.

        - **Why Web Components Are So Important**: Leon Revill compares web components with concepts from different frameworks and explains why web components matter.
        - **Write Web Components with ES2015 (ES6)**: This tutorial shows how to create a web component using ES2015 and how to make use of babel to transpile back to ES5.
- **DOM Diffing & Patching**: Diffing & Patching is a pattern which allows faster and simpler rendering and updating of DOM trees as manual manipulation à la jQuery.
    - **Change And Its Detection In JavaScript Frameworks**: This article explores several approaches to manage state: Ember's data binding, Angular's dirty checking, React's virtual DOM, and its relationship to immutable data structures.
    - **Cito.js**: The core of cito.js consists of a virtual DOM library inspired by React/Mithril. On top of that, it will provide a component framework which will make it easy to build well-encapsulated components.
    - **Incremental DOM**: Incremental DOM is a library for building up DOM trees and updating them in-place when data changes. It differs from the established virtual DOM approach in that no intermediate tree is created (the existing tree is mutated in-place).
        - **Introducing Incremental DOM**: Incremental DOM is a library inspired by Virtual DOM developed at Google.
    - **Morphdom**: Lightweight module for morphing an existing DOM node tree to match a target DOM node tree. It's fast and works with the real DOM—no virtual DOM here!
    - **React Demystified**: This article is an attempt to explain the core ideas behind React.js and Virtual DOM.
    - **React vs Incremental DOM vs Glimmer**: In this post we will explore three technologies to build dynamic DOMs. We will also run benchmarks and find out which one is faster.
    - **React-less Virtual DOM with Snabbdom: functions everywhere!**: Yassine Elouafi shows in this post how to write a virtual DOM based applications using a small and standalone library.
    - **Snabbdom**: A virtual DOM library with focus on simplicity, modularity, powerful features and performance.
    - **Virtual DOM**: Virtual-dom is a collection of modules designed to provide a declarative way of representing the DOM for your app. So instead of updating the DOM, you simply create a virtual tree or VTree, which looks like the DOM state that you want.
        - **html-to-vdom**: This is yet another library to convert HTML into a vtree. It's used in conjunction with virtual-dom to convert template based views into virtual-dom views.
        - **html2hyperscript**: Automatically translate old HTML markup into the new Hyperscript markup embeddable directly inside your component Javascript code.
        - **vdom-to-html**: Turn Virtual DOM nodes into HTML.
        - **vdom-virtualize**: Turn a DOMNode into a virtual-dom node.
        - **virtual-html**: Convert given HTML into Virtual DOM object.

        - **vtree-select**: Select vtree nodes (used by virtual-dom) using css selectors. Selector matching is done using cssauron. See the documentation for details on supported selectors.
  - **Design Pattern Collections**: Overview resources and collections of design patterns.
    - **About HTML Semantics and Frontend Architecture**: A collection of thoughts, experiences, ideas on HTML semantics, components and approaches to front-end architecture, class naming patterns, and HTTP compression.
    - **Box Tech Talk: Scalable JavaScript Application Architecture**: A video by Nicholas Zakas (2012) about JavaScript Architecture.
    - **Learning JavaScript Design Patterns**: In this free book Addy Osmani explores applying both classical and modern design patterns to the JavaScript programming language.
    - **Patterns For Large-Scale JavaScript Application Architecture**: An extensive overview by Addy Osmani of existing architectural solutions in the frontend development field.
    - **Scalable JavaScript Application Architecture**: In this video (2011) Nicholas Zakas discusses frontend architecture for complex, modular web applications with significant JavaScript elements.
    - **Single Page Apps in Depth**: This free book is what I wanted when I started working with single page apps. It's not an API reference on a particular framework, rather, the focus is on discussing patterns, implementation choices and decent practices.
  - **JavaScript Modules**: Modules divide programs into clusters of code that, by some criterion, belong together.
    - **Chapter 10 of Eloquent JavaScript: Modules**: This chapter explores some of the benefits that division of code provides and shows techniques for building modules in JavaScript.
    - **ES6 In Depth: Modules**: This article highlights export and import keywords from ES6.
    - **Efficient Module Loading Without Bundling**: We can combine ES2015 modules, static analysis of those modules, HTTP/2, caching, Service Workers and a bloom-filter to create a server-client relationship where the client can efficiently load any module.
    - **JavaScript Modules: A Beginner’s Guide**: In this post, Preethi Kasireddy will unpack the buzzwords like module bundlers, AMD and CommonJS for you in plain English, including a few code samples.
    - **Modern Modular JavaScript Design Patterns**: A chapter from Essential JavaScript Design Patterns on Modules.
    - **Module Bundlers and Loaders**: Libraries for bundling JavaScript Modules into one or several files.
      - **Browserify**: Browserify lets you require('modules') in the browser by bundling up all of your dependencies.

    - **Budo**: A browserify development server, focused on incremental reloading, LiveReload integration (including CSS injection), and other high-level features.
    - **Watchify**: Watch mode for browserify builds.
- **CrapLoader**: The goal of crapLoader is to load ads, widgets or any JavaScript code with document.write in it. This library hijacks document.write and delegates the content loaded from each script into the correct position.
- **Modules Webmake**: A CommonJS module bundler similar to Browserify but much faster due to different requirements finder.
- **Require.js**: RequireJS is a JavaScript file and AMD module loader. It is optimized for in-browser use, but it can be used in other JavaScript environments.
- **Require1k**: CommonJS require for the browser in 1KB, with no build needed.
- **Rollup.js**: Rollup is a next-generation JavaScript module bundler. Author your app or library using ES2015 modules, then efficiently bundle them up into a single file for use in browsers and Node.js.
- **SystemJS**: Universal dynamic module loader - loads ES6 modules, AMD, CommonJS and global scripts in the browser and NodeJS. Works with both Traceur and Babel.
    - **Modular JavaScript: A Beginners Guide to SystemJS & JSPM**: The combination of jspm and SystemJS provides a unified way of installing and loading dependencies.
- **URequire**: The Ultimate JavaScript Module Builder & Automagical Task Runner.
- **Webpack**: Webpack is a module bundler. It takes modules with dependencies and generates static assets representing those modules.
    - **Block, Element, Modifying Your JavaScript Components**: Mark Dalgleish is discussing how to organize React code with BEM and build everything with Webpack.
    - **Developing with Docker and Webpack**: Chris Harrington explains how to create a development environment with Webpack and Docker to match the production as much as possible.
    - **Full-Stack Redux Tutorial**: We will go through all the steps of constructing a Node+Redux backend and a React+Redux frontend for a real-world application, using test-first development.
    - **How to Set Up Webpack Image Loader**: This brief tutorial will help you set up an image loader in Webpack.
    - **The SoundCloud Client in React + Redux**: After finishing this step by step tutorial you will be able to author your own React + Redux project with Webpack and Babel.
    - **Webpack from Apprentice to Master**: The purpose of this guide is to help you get started with Webpack and then go beyond basics.

        - **WebpackBin**: A webpack code sandbox.
        - **Why I think Webpack is the Right Approach To Build Pipelines**: Thomas Boyt compares how Grunt, Gulp, Broccoli and Webpack discover dependencies.
    - **UMD (Universal Module Definition)**: This repository formalizes the design and implementation of the Universal Module Definition (UMD) API for JavaScript modules. These are modules which are capable of working everywhere, be it in the client, on the server or elsewhere.
    - **Writing Modular JavaScript With AMD, CommonJS & ES Harmony**: In this article Addy Osmani reviewes several of the options available for writing modular JavaScript using modern module formats AMD, CommonJS and ES6 Modules.
- **Observable**: An Observable is an event stream which can emit zero or more events, and may or may not finish. If it finishes, then it does so by either emitting an error or a special “complete” event.
    - **ECMAScript Observable**: This proposal introduces an Observable type to the ECMAScript standard library. The Observable type can be used to model push-based data sources such as DOM events, timer intervals, and sockets.
    - **Reactive Extensions (RxJS)**: RxJS is a set of libraries for composing asynchronous and event-based programs using observable sequences and fluent query operators.
        - **Async JavaScript with Reactive Extensions**: Jafar Husain explains in this video how Netflix uses the Reactive Extensions (Rx) library to build responsive user experiences that strive to be event-driven, scalable and resilient.
        - **Exploring Rx Operators: FlatMap**: Christoph Burgdorf introduces the FlatMap operator and its usage for collections and observables.
        - **Exploring Rx Operators: Map**: Christoph Burgdorf explains how to use the map operator in RxJS.
        - **Functional Core Reactive Shell**: Giovanni Lodi makes an overview of different architecture meta-patterns and describes his current findings about functional programming and observables as a way to control side effects.
        - **Learn RX**: A series of interactive exercises for learning Microsoft's Reactive Extensions (Rx) Library for Javascript.
        - **Learn RxJS**: This site focuses on making RxJS concepts approachable, the examples clear and easy to explore, and features references throughout to the best RxJS related material on the web.
        - **Real World Observables**: Sergi Mansilla writes an FTP client to use it as an example for a real world application based on RxJS.
        - **Rx Training Games**: Rx Training Games is a coding playground that can be used to learn and practice Reactive Extensions coding grid-based games
        - **Rx-Book**: A complete book about RxJS v.4.0.

        - **RxMarbles**: A webapp for experimenting with diagrams of Rx Observables, for learning purposes.
        - **RxState**: Simple opinionated state management library based on RxJS and Immutable.js
        - **Taking Advantage of Observables in Angular 2**: Christoph Burgdorf describes the advantages of Observables and how you can use them in Angular 2 context.
        - **Transducers with Observable Sequences**: A chapter from the RxJS Book describing Transducers.
        - **Why We Built Xstream**: The authors needed a stream library tailored for Cycle.js. It needs to be “hot” only, small in kB size and it should have only a few and intuitive operators.
  - **Routing**: A routing system parses a string input (usually a URL) and decides which action should be executed by matching the string against multiple patterns.
    - **A JavaScript router in 20 lines**: Joakim Carlstein shows how to write a simple router with data binding.
    - **Crossroads.js**: Crossroads.js is a powerful and flexible routing system. If used properly it can reduce code complexity by decoupling objects and also by abstracting navigation paths and server requests.
    - **Director**: A tiny and isomorphic URL router for JavaScript.
    - **Encapsulated Routing with Elements**: Peter Burns describes a routing approach based on Polymer elements, that allow to create chained and modular routes.
    - **Hash.js**: Hash.js is a 0.5 KB script that lets you manipulate everything behind # in urls.
    - **JQuery Address**: The jQuery Address plugin provides powerful deep linking capabilities and allows the creation of unique virtual addresses that can point to a website section or an application state.
    - **Page.js**: Micro client-side router inspired by the Express router.
    - **Roadcrew.js**: Roadcrew.js is a small JavaScript component which lets you switch pages of a single file website.
    - **Route Recognizer**: A lightweight JavaScript library that matches paths against registered routes. It includes support for dynamic and star segments and nested handlers.
    - **Router.js (Ember)**: Router.js is the routing microlib used by Ember.js.
    - **Router5**: A simple, powerful, modular and extensible router, organising your named routes in a tree and handling route transitions. In its simplest form, Router5 processes routing instructions and outputs state updates.
  - **UI Data Binding**: Binding of UI elements to an application domain model. Most frameworks employ the Observer pattern as the underlying binding mechanism.
    - **Bindings in Ember**: Unlike most other frameworks that include some sort of binding implementation, bindings in Ember.js can be used with any object.

        - **Change And Its Detection In JavaScript Frameworks**: This article explores several approaches to manage state: Ember's data binding, Angular's dirty checking, React's virtual DOM, and its relationship to immutable data structures.
        - **Easy Two-Way Data Binding in JavaScript**: Two-way data binding refers to the ability to bind changes to an object's properties to changes in the UI, and viceversa. This article describes how to implement data binding with vanilla JavaScript.
        - **Functional Reactive Bindings**: A CommonJS package that includes functional and generic building blocks to help incrementally ensure consistent state.
        - **Knockout.js**: Knockout is a standalone JavaScript implementation of the Model-View-ViewModel pattern with templates.
        - **Rivets.js**: Lightweight and powerful data binding + templating solution for building modern web applications.
        - **Synapse**: Hooks to support data binding between virtually any object.
    - **Unidirectional Data Flow**: An architecture design pattern which promotes a flow of data and events in a single direction, usually creating an interactive loop.
        - **Flux**: Flux is the application architecture that Facebook uses for building client-side web applications. It complements React's composable view components by utilizing a unidirectional data flow. It's more of a pattern rather than a formal framework, and you can start using Flux immediately without a lot of new code.
            - **Fluxiny**: ~1K implementation of flux architecture
        - **Immutable User Interfaces**: Lee Byron talks about unidirectional data flow architectures based on immutable data structures in contrast to traditional MVC based designs.
            - **Immutable.js**: Immutable persistent data collections for Javascript which increase efficiency and simplicity.
        - **MobX**: MobX is a battle tested library that makes state management simple and scalable by transparently applying functional reactive programming.
        - **Model-View-Intent (MVI)**: MVI is a unidirectional data flow architecture pattern consisting of three parts: Intent (to listen to the user), Model (to process information), and View (to output back to the user).
            - **MVI in Cycle.js Docs**: André Staltz describes how to refactor an application into MVI pattern.
            - **Model-View-Intent with React and RxJS**: Satish Chilukuri shows an example implementation of MVI pattern with React.
            - **Reactive MVC and the Virtual DOM**: André Staltz describes the idea of Reactive Programming vs. Interactive Programming, proceeds with the MVI design pattern and compares it to React/Flux.
            - **What Developers Need to Know about MVI (Model-View-Intent)**: The article explains the general MVI pattern and how it relates to React, Reactive Programming and Cycle.js

- **Nothing New in React and Flux Except One Thing**: Andre Staltz talks about aspects of React and Flux which make them innovative and compelling.
- **Redux**: Redux is a predictable state container for JavaScript apps. It attempts to make state mutations predictable by imposing certain restrictions on how and when updates can happen.
    - **Building Redux in TypeScript with Angular 2**: In this post we’re going to discuss the ideas behind Redux. How to build our own mini version of the Redux Store and hook it up to Angular 2.
    - **Exploring Redux Middleware**: The author explains how to author your own middleware for Redux. He dives into the execution path of each middleware function in the chain and shows some examples.
    - **Full-Stack Redux Tutorial**: We will go through all the steps of constructing a Node+Redux backend and a React+Redux frontend for a real-world application, using test-first development.
    - **Immutable.js**: Immutable persistent data collections for Javascript which increase efficiency and simplicity.
    - **Learn Redux**: A video series by Wes Bos, teaching Redux. From setting up Webpack to using Dev Tools.
    - **Normalizr**: Normalizes deeply nested JSON API responses according to a schema for Flux and Redux apps.
    - **Redux Actions**: Flux Standard Action utilities for Redux.
    - **Redux Form**: A Higher Order Component using react-redux to keep form state in a Redux store.
    - **Redux Loop**: A port of elm-effects and the Elm Architecture to Redux that allows you to sequence your effects naturally and purely by returning them from your reducers.
    - **Redux Saga**: An alternative Side Effects middleware for Redux applications. Instead of dispatching Thunks which get handled by the redux-thunk middleware, you create Sagas to gather all your Side Effects logic in a central place.
    - **Redux Tutorial**: This repository contains a step by step tutorial to help grasp flux and more specifically Redux.
    - **Reinventing Flux - Interview with Dan Abramov**: Dan talks about why he developed Redux.
    - **Reselect**: Simple “selector” library for Redux inspired by getters in NuclearJS and subscriptions in re-frame.
    - **Some Problems with React/Redux**: André Staltz goes through the pros and cons of React + Redux.
    - **Testing a React & Redux Codebase**: This series aims to be a very comprehensive guide through testing a React and Redux codebase, where you can really cover a

lot with just unit tests because the code is mostly universal.
        - **The Redux Ecosystem**: Let’s take a look at most of the features that you’ll have to deal with when the time comes,—and where React & Redux themselves can’t help you.
        - **The Redux Journey at react-europe 2016**: In this talk, Dan Abramov reflects on the past, present, and future of Redux.
        - **The SoundCloud Client in React + Redux**: After finishing this step by step tutorial you will be able to author your own React + Redux project with Webpack and Babel.
        - **Tic-Tac-Toe.js: Redux Pattern in Plain JavaScript**: Ramon Victor describes how to use Redux with vanilla JavaScript. No React, no jQuery, no micro-library, it doesn't rely on anything else. It's just plain JS.
        - **Understanding Redux Middleware**: The author describes the functional programming concepts involved in the creation and application of middleware functions.
    - **Unidirectional Data Flow Architectures (Talk)**: Andre Staltz compares modern architecture patterns including Flux, Redux, Model-View-Intent, Elm Arch and BEST.
- **Designs**: Ready to use and well documented structures and frameworks for frontend development.
  - **Atomic Design**: Atomic Design discusses the importance of crafting robust design systems, and introduces a methodology for which to create smart, deliberate interface systems.
    - **A More Seamless Workflow—Style Guides for Better Design and Development**: Ash Connolly explains what styles guides are and which benefits they bring to designers and developers.
    - **Atomic Docs**: Atomic Docs is a styleguide generator and component manager. Atomic Docs is built in PHP. Inspired by Brad Frost's Atomic Design principles.
    - **Atomic Lab**: Template sharing and coding environment based on atomic design.
  - **Authoring jQuery Plugins**: jQuery is an utility library and a plugin framework. This section collects resources about creating such plugins.
    - **Advanced Plugin Concepts**: A collection of best practices for jQuery plugin authoring.
    - **How to Create a Basic Plugin**: The article describes basic plugin creation and provides a simple boilerplate.
    - **Signs of a poorly written jQuery plugin**: Collection of jQuery plugin antipatterns.
    - **The Ultimate Guide to Writing jQuery Plugins**: A comprehensive guide on how to develop jQuery plugins including a simple boilerplate.
    - **Writing Stateful Plugins with the jQuery UI Widget Factory**: The article demonstrates the capabilities of the Widget Factory by building a simple progress bar plugin.
    - **jQuery Boilerplate**: This project won't seek to provide a perfect solution to every possible pattern, but will attempt to cover a simple template for beginners and above.

    - **jQuery Plugin Patterns**: This project won't seek to provide implementations for every possible pattern, but will attempt to cover popular patterns developers often use in the wild.
  - **Block Element Modifier (BEM)**: Methodology aimed at achieving fast to develop long-lived projects, team scalability, and code reuse.
    - **A New Front-End Methodology: BEM**: An introduction by Varvara Stepanova at SmashingMagazine.
    - **An Introduction to the BEM Methodology**: General introduction article on tutsplus.
    - **BEM 101**: A collaborative post by Joe Richardson, Robin Rendle, and CSS-Tricks staff giving an introduction to BEM with some good examples.
    - **BEM I (finally) understand**: In this article Andrei Popa will focus on the basics of BEM and how to approach simple to complex anatomies.
    - **Battling BEM (Extended Edition): 10 Common Problems And How To Avoid Them**: This article aims to be useful for people who are already BEM enthusiasts and wish to use it more effectively or people who are curious to learn more about it.
    - **Block, Element, Modifying Your JavaScript Components**: Mark Dalgleish is discussing how to organize React code with BEM and build everything with Webpack.
    - **Emmet filter for BEM**: If you’re writing your HTML and CSS code in OOCSS-style, Yandex’s BEM style specifically, you will like this filter. It provides some aliases and automatic insertions of common block and element names in classes.
    - **Fifty Shades of BEM**: Article describes different flavors of BEM.
    - **How We Use BEM to Modularise Our CSS**: Andrei Popa describes the challenges, AlphaSights team had, implementing BEM in their projects.
    - **Introduction To BEM Methodology (Toptal)**: General introduction to BEM methodology and platform.
    - **MindBEMding – getting your head ’round BEM syntax**: Article on csswizardry explaining the BEM syntax for CSS classes.
    - **Pobem**: PostCSS plugin for BEM syntax.
    - **Support for BEM modules in Sass 3.3**: The next major release of Sass is poised for release and with it comes real support for BEM-style modules...
    - **To BEM or not to BEM**: A series of interviews on BEM methodology.
  - **Cycle.js**: A functional and reactive JavaScript framework that solves the cyclic dependency of Observables which emerge during dialogues (mutual observations) between the Human and the Computer.
    - **Async Driver**: Higher order factory for creating cycle.js async request based drivers. Allows you almost completely eliminate boilerplate code for this kind of drivers.
    - **Cycle.js Was Built to Solve Problems**: In this video André Staltz shows how Cycle.js has a practical purpose, meant to solve problems your customers/business may relate to.

- **Cycle.js and Functional Reactive User Interfaces**: In this talk we will discover how Cycle.js is purely reactive and functional, and why it's an interesting alternative to React.
- **Draw Cycle**: Simple Cycle.js program visualized
- **Drivers**: Drivers are functions that listen to Observable sinks (their input), perform imperative side effects, and may return Observable sources (their output).
  - **Animation**: A Cycle driver for requestAnimationFrame.
  - **Audio Graph Driver**: Audio graph driver for Cycle.js based on virtual-audio-graph.
  - **Cookie**: Cycle.js Cookie Driver, based on cookie_js library.
  - **DOM**: The standard DOM Driver for Cycle.js based on virtual-dom, and other helpers.
  - **Fetch**: A Cycle.js Driver for making HTTP requests, using the Fetch API.
  - **Fetcher**: A Cycle.js Driver for making HTTP requests using stackable-fetcher.
  - **Firebase**: Thin layer around the firebase javascript API that allows you to query and declaratively update your favorite real-time database.
  - **HTTP**: A Cycle.js Driver for making HTTP requests, based on superagent.
  - **Hammer.js**: The driver incorporates the Hammer.js gesture library.
  - **History**: Cycle.js URL Driver based on the rackt/history library.
  - **Keys**: A Cycle.js driver for keyboard events.
  - **Mongoose.js**: A driver for using Mongoose with Cycle JS. Accepts both, write and read operations.
  - **Notification**: A Cycle.js Driver for showing and responding to HTML5 Notifications.
  - **Router**: A router built from the ground up with Cycle.js in mind. Stands on the shoulders of battle-tested libraries switch-path for route matching and rackt/history for dealing with the History API.
  - **Router5**: A source/sink router driver for Cycle.js, based on router5.
  - **Server-Sent Events**: Cycle.js driver for Server-Sent Events (SSE), a browser feature also known as EventSource. Server-Sent Events allow the server to continuously update the page with new events, without resorting to hacks like long-polling.
  - **Snabbdom**: Alternative DOM driver utilizing the snabbdom library.
  - **Socket.IO**: A Cycle driver for applications using Socket.IO
  - **Storage**: A Cycle.js Driver for using localStorage and sessionStorage in the browser.
- **Example Projects**: Example applications built with Cycle.js
  - **Cycle.js Examples**: Browse and learn from examples of small Cycle.js apps using Core, DOM Driver, HTML Driver, HTTP Driver, JSONP Driver, and others.
  - **RX Marbles**: Interactive diagrams of Rx Observables.
  - **TODO: Minimum Viable Pizza**: Minimum Viable Pizza implemented with Cycle.js
  - **Tricycle**: A scratchpad for trying out Cycle.js.

        - **Intro to Functional Reactive Programming with Cycle.js**: Nick Johnstone gives an introduction to developing with Cycle.js in this video presentation.
        - **Learning How to Ride: an Introduction to Cycle.js**: In this talk, Fernando Macias Pereznieto introduces us to the good, the bad, and the beautiful of using Cycle.js, whether you are a complete beginner or an experienced JS ninja.
        - **Motorcycle.js**: This is a sister project that will continue to evolve and grow alongside Cycle.js for the foreseeable future. The primary focus of this project is to tune it for performance as much as possible.
            - **Most**: Monadic reactive streams with high performance.
        - **Plug and Play All Your Observable Streams With Cycle.js**: Frederik Krautwald explains the principles behind Cycle.js, it's inner workings and how to use it to create a simple program with drivers.
        - **Tricycle**: A scratchpad for trying out Cycle.js.
        - **What Developers Need to Know about MVI (Model-View-Intent)**: The article explains the general MVI pattern and how it relates to React, Reactive Programming and Cycle.js
    - **Polymer Project**: The Polymer library is designed to make it easier and faster for developers to create great, reusable components for the modern web.
        - **Building web components using ES6 classes**: Web components evolve markup into something that's meaningful, maintainable, and highly modular. Thanks to these new API primitives, not only do we have improved ergonomics when building apps, but we gain better overall structure, design, and reusability.
        - **Developing Front-End Microservices**: In this article series we'll go through Web Components development in context of microservices.
        - **Lazy Loading of Pages**: iron-lazy-pages is a Polymer component which allows to load pages on demand.
        - **ShadowDOM Visualizer**: This tools allows you to visualize how Shadow DOM renders in the browser.
        - **Thinking in Polymer (The Polymer Summit 2015)**: Kevin Schaaf explains how to employ data binding and composition to build complex application using only Polymer.
        - **Unidirectional Dataflow Architecture with Polymer + RxJS + Immutable Data**: Richard Anaya describes how to combine Polymer, RxJS and Freezer to implement a unidirectional data flow architecture.
        - **Using Elements**: This guide describes how to install and use standalone Polymer components in an existing project.
        - **Using Polymer with Flux and a Global App State**: Paulus Schoutsen describes his experience integrating Polymer and NuclearJS.
        - **What is shady DOM?**: On browsers that support shadow DOM, it's possible to have an element that is rendered with complex DOM, but have that complexity hidden away as implementation detail.

    - **SMACSS**: SMACSS (pronounced “smacks”) is a way to examine your design process and as a way to fit those rigid frameworks into a flexible thought process. It is an attempt to document a consistent approach to site development when using CSS.
    - **T3**: T3 is a minimalist JavaScript framework sponsored by Box Inc. that provides core structure to code.
    - **The Elm Architecture**: The Elm Architecture is a simple pattern for infinitely nestable components. It is great for modularity, code reuse, and testing.
    - **TodoMVC**: A project which offers the same Todo application implemented using MV* concepts in most of the popular JavaScript MV* frameworks of today.
- **Event-Driven Programming**: Event-driven programming is a programming paradigm in which the flow of the program is determined by events such as user actions, sensor outputs, or messages from other programs/threads.
    - **Comparison Between Different Observer Pattern Implementations**: The comparison below is just about the basic features of subscribing to an event type, dispatching and removing an event listener.
    - **Event Emitter, Pub Sub or Deferred Promises**: In this post Pete Otaqui explores a little about how each pattern works with (very) basic implementations and looks at the reasons why you might choose one over another.
    - **Implementations**: Libraries, frameworks and tools that use Event-Driven Programming paradigm.
        - **Bacon.js**: A small functional reactive programming lib for JavaScript. Turns your event spaghetti into clean and declarative feng shui bacon, by switching from imperative to functional.
        - **Flight**: An event-driven web framework, from Twitter.
        - **Mediator.js**: Mediator is a simple class that allows you to register, unregister, and call subscriber methods to help event-based, asyncronous programming.
        - **Postal.js**: Postal.js takes the familiar "eventing-style" paradigm and extends it by providing "broker" and subscriber implementations
        - **Radio.js**: Radio.js is a small dependency-free publish/subscribe JavaScript library. Use it to implement the observer pattern in your code to help decouple your application architecture for greater maintainability.
        - **js-signals**: Custom Event/Messaging system for JavaScript.
        - **pubsub.js**: A tiny (~600 bytes when minified, ~300 bytes when gzip'd) and robust pubsub implementation.
- **Functional Programming**: Functional programming is a programming paradigm, that treats computation as the evaluation of mathematical functions and avoids changing-state and mutable data.
    - **A Gentle Introduction to Functional JavaScript**: A 3 part series, by Derick Bailey featuring Chet Harrison, about functional programming with many examples in JavaScript.

    - **Monads, Monoids and Composition with Functional JavaScript**: Chet Harrison explains monads using form validation as an example.
    - **Notes and Code from the Crowdcast**: Chet Harrison provides a broad overview of functional programming concepts and a step by step tutorial for building Monads.
    - **The Basics of Functional Programming**: In this first episode, you'll learn the basics of why functional programming, what it is, where it came from and what the core of it is. You'll see function composition, function purity, currying, higher order functions and first-class functions.
- **A Million Ways to Fold in JS**: Brian Lonsdorf provides many functional alternatives to loops in this video.
- **Adventures in Functional Programming**: A talk by Jim Weirich, demonstrating how to use functional programming and lambda calculus to derive Y combinator.
- **Allong.es**: allong.es is a JavaScript library based on the function combinator and decorator recipes introduced in the book JavaScript Allongé.
- **Barely Functional**: Tiny (2.7kb) functional programming library using native es5/6 operations.
- **Basic Lazy Evaluation and Memoization in JavaScript**: Memoization is a way of optimizing code so that it will return cached results for the same inputs.
- **Bilby.js**: A functional library based on category theory with immutable multimethods, functional data structures, functional operator overloading, automated specification testing.
- **Composability: from Callbacks to Categories in ES6**: The author borrows some ideas from functional languages to explore a different approach for addressing the callback hell.
- **Curry or Partial Application?**: Eric Elliott describes the difference between partial application and curry.
- **Daggy**: Library for creating tagged constructors with catamorphism.
- **Date FP**: Functional programming date manipulation library.
- **Debugging Functional**: This post will demonstrate a simple solution that can go a long way to enhance the debugging experience in functional JavaScript applications.
- **Deterministic**: A weekly digest of interesting news and articles covering functional programming for the web, especially on the front end.
- **Example Projects**: Open source projects which use functional programming, preferably point-free and side-effect-free.
    - **Async Problem**: This project considers various approaches to the problem of concurrently reading files inside a directory and concatenating their contents.
    - **CommonJS module dependencies resolver**: The module and all related modules are written using point-free style.
    - **Egg Hunt Server**: A restful API written in FP style.
    - **Idealist**: Functional HTTP micro-framework.
    - **Sanctuary Build Script**: A build script for generating the Sanctuary website.

- **FP DOM**: A collection of functions to favor functional programming in a DOM context.
- **Fantasy Combinators**: Combinators which are used for fantasy-land projects.
- **Fantasy Land**: Specification for interoperability of common algebraic structures in JavaScript.
  - **Conformant Implementations**: A list of libraries implementing the Fantasy Land specification.
- **Fantasy Lenses**: Composable, immutable getters and setters.
- **Functional Concepts For JavaScript Developers: Currying**: Andrew Robbins talks about what currying is and why it's useful.
- **Functional Core Reactive Shell**: Giovanni Lodi makes an overview of different architecture meta-patterns and describes his current findings about functional programming and observables as a way to control side effects.
- **Functional Frontend Architecture**: This repository is meant to document and explore the implementation of what is known as "the Elm architecture". A simple functional architecture for building frontend applications.
- **Functional JavaScript Mini Book**: Jichao Ouyang gives and introduction to functional programming with JavaScript and describes some Typeclasses like Functor and Monad.
- **Functional Javascript Workshop**: The goal of this workshop is to create realistic problems that can be solved using terse, vanilla, idiomatic JavaScript.
- **Functional Principles In React**: Jessica Kerr talks about four functional principles: Composition, Declarative Style, Isolation and Flow Of Data, and their usage in React.
- **Functional Programming Jargon**: Jargon from the functional programming world in simple terms.
- **Functional Programming for JavaScript People**: Chet Corcos explains different features of functional programming like composition, currying, lazy evaluation, referential transparency and compares Clojure with Haskell.
- **Functional Refactoring in JavaScript**: In this article Victor Savkin shows how to apply functional thinking when refactoring JavaScript code. He does that by taking a simple function and transforming it into a more extendable one, which has no mutable state, and no if statements.
- **Functional.js**: Functional.js is a functional JavaScript library. It facilitates currying and point-free / tacit programming and this methodology has been adhered to from the ground up.
- **Functionize**: A collection of functions which aids in making non-functional libraries functional.
- **Futures and Monoids**: Yassine Elouafi explains the nature of Monoids using Futures, Numbers and Strings as examples.
- **Hey Underscore, You're Doing It Wrong!**: In this talk Brian Lonsdorf gently takes a shot at underscore.js for not thinking about currying and partial function application in its library design.

- **Immutability, Interactivity & JavaScript**: We'll dive in and see how trees of JavaScript arrays can permit building efficient immutable collections. Then we'll see how embracing immutable values dramatically simplifies some classic hard problems in client side programming including but not limited to undo, error playback, and online/offline synchronization.
- **Immutable Sequence.js**: High performance implementation of Immutable Sequence in JavaScript, based on Finger Tree.
- **Immutable.js**: Immutable persistent data collections for Javascript which increase efficiency and simplicity.
- **JSAir - Functional and Immutable Design Patterns in JavaScript**: An episode of JavaScript Air about "the how and why of functional programming and immutable design patterns in JavaScript" with Dab Abramov and Brian Lonsdorf as guests.
- **JavaScript and Type Thinking**: Yassine Elouafi introduces Algebraic Data Types with an example of a simple and a recursive type.
- **Javascript Combinators by Reginald Braithwaite**: In this talk, we'll explore functions that consume and return functions, and see how they can be used to build expressive programs that hew closely to JavaScript's natural style.
- **Lamda.js**: This library takes all the methods on instances of strings, arrays, objects, numbers, and regexp's and turns them into functions that can be used in a pointfree way.
- **Lenses Quick n' Dirty**: A video by Brian Lonsdorf that introduces lenses.
- **Lenses and Virtual DOM Support Open Closed**: Hardy Jones explains how Lenses work using a simple example of working with Virtual DOM.
- **Lenses.js**: Composable kmett style lenses.
- **Lodash/fp**: The lodash/fp module is an instance of lodash with its methods wrapped to produce immutable auto-curried iteratee-first data-last methods.
- **Making your JavaScript Pure**: Jack Franklin compares pure and impure functions and describes how to leverage functional programming principles in JavaScript.
- **Monads**: Composable computation descriptions. The essence of monad is thus separation of composition timeline from the composed computation's execution timeline, as well as the ability of computation to implicitly carry extra data.
  - **Collections of Monads**: Libraries of monad implementations.
    - **Akh**: Akh includes a basic set of common monad transformers, along with monads derived from these transformers. Akh structures implement the Fantasy Land specification.
    - **Folktale**: Folktale is a suite of libraries for generic functional programming in JavaScript that allows you to write elegant modular applications with fewer bugs, and more reuse.
    - **Monet.js**: Monet is a tool bag that assists Functional Programming by providing a rich set of Monads and other useful functions.

- **Continuation Monad**: Represents computations in continuation-passing style (CPS). In continuation-passing style function result is not returned, but instead is passed to another function, received as a parameter (continuation).
  - **The Delimited Continuation Monad in Javascript**: This post overviews continuations in Atum and covers the implementation of the delimited continuation monad in JavaScript.
- **Either Monad**: The Either type represents values with two possibilities: a value of type Either a b is either Left a or Right b. It is often used for error handling.
  - **Lazy Either**: The LazyEither type is used to represent a lazy Either value. It is similar to the Future and Promise types.
  - **Practical Intro to Monads in JavaScript: Either**: Jakub Strojewski describes the Either Monad, a tool for fast-failing, synchronous computation chains.
- **Free Monad**: A free monad satisfies all the Monad laws, but does not do any computation. It just builds up a nested series of contexts. The user who creates such a free monadic value is responsible for doing something with those nested contexts.
  - **Fantasy Frees**: An implementation of Coyoneda, Yoneda, Trampoline, Free Monad and Free Applicative with usage examples.
  - **Free Monad Experiments by Hardy Jones**: Coyoneda, Coproduct, Either, Free, State, AJAX and so on.
  - **Free Monads Video Series**: A video series on free monads by Brian Lonsdorf explaining Coyoneda, Free Monad and Interpretors.
  - **Freeky**: Collection of free monads by Brian Lonsdorf.
- **Futures**: Futures represent the value arising from the success or failure of an asynchronous operation (I/O).
  - **Fluture**: The debuggable Fantasy Land Future library.
  - **Folktale Task**: A structure for time-dependent values, providing explicit effects for delayed computations, latency, etc.
  - **From Callback to Future -> Functor -> Monad**: Yassine Elouafi goes through a simple implementation of Futures and compares them to Promises.
  - **Future IO**: A fantasy-land compliant monadic IO library for Node.js.
  - **Futurizer**: Turn callback-style functions or promises into futures!
- **Introduction**: Introductory materials about monads.
  - **Monads in JavaScript**: This article explains monads and their usage in JavaScript including Identity, Maybe, List, Continuation, Do notation and Chaining.
  - **Practical Intro to Monads in JavaScript**: A simple, practical tutorial for JavaScript developers showing how some monads can be used.
  - **Understanding Monads With JavaScript**: The author starts with a problem of dealing with explicit immutable state and solves it with JavaScript using monads.

    - **Maybe Monad**: Using Maybe is a good way to deal with errors or exceptional cases without resorting to drastic measures such as error. It is a simple kind of error monad, where all errors are represented by Nothing. A richer error monad can be built using the Either type.
        - **A Gentle Intro to Monads … Maybe?**: A short introduction to Maybe and the world of monads.
        - **A Monad in Practicality: First-Class Failures**: This article shows how the Maybe monad can be used for handling simple failure use cases. It then extrapolates into complex failure scenarios and shows how these cases can be modelled in terms of the Either monad.
        - **Practical Intro to Monads in JavaScript**: A simple, practical tutorial for JavaScript developers showing how some monads can be used.
    - **Reader Monad**: Represents a computation, which can read values from a shared environment, pass values from function to function, and execute sub-computations in a modified environment.
        - **Don’t Fear the Reader**: Pascal Hartig explains how to use the reader monad in JavaScript.
        - **Fantasy Readers**: Fantasy Land compatible implementation of the Reader Monad.
        - **LiveCoding Video of Reader Monad Implementation**: In this video you will learn how to use and implement a Reader from scratch.
        - **Monad a Day: Reader**: Short video by Brian Lonsdorf about the Reader Monad.
    - **Transformers**: Special types that allow us to roll two monads into a single one that shares the behavior of both.
        - **Akh**: Akh includes a basic set of common monad transformers, along with monads derived from these transformers. Akh structures implement the Fantasy Land specification.
        - **Fantasy ArrayT**: Monad transformer for JavaScript Arrays.
        - **Monad Transformers**: Monad transformers are tricky, they require an excessive amount of type juggling. One of the aims of this package is to reduce the amount of wrapping and unwrapping needed for making a new transformer and to provide an easy way to define and combine transformers.
        - **Monad Transformers Library**: Practical monad transformers for JS.
    - **Validation Monad**: A disjunction that is appropriate for validating inputs and aggregating failures.
        - **Folktale Validation**: Validation Monad implementation of Folktale Library.
        - **Practical Intro to Monads in JavaScript: Validation**: Jakub Strojewski shows how to accumulate errors in a simple Validation use case.
- **Mori**: A library for using ClojureScript's persistent data structures and supporting API from the comfort of vanilla JavaScript.

- **Mostly Adequate Guide to Functional Programming**: A book by Brian Lonsdorf that introduces algebraic functional programming in JavaScript.
- **Nanoscope**: Nanoscope is a javascript library designed to make complex transformations of data much easier. It is a built on the idea of a functional Lens - a construct that enables focusing on sub-parts of data structures to get and modify.
- **Pointfree Fantasy**: Point-free wrappers for fantasy-land. Functions are curried using lodash's curry function, and receive their data last. Gives us aliases with our familar haskell names as well.
- **Pointfree Javascript**: In this post Lucas Reis presents what is called pointfree style programming and goes through some common scenarios to demonstrate its benefits.
- **Practical Functional Programming: Pick Two**: James Coglan tries to show in this video how to use functional concepts in daily JavaScript programming.
- **Promises + FP = Beautiful Streams**: Yassine Elouafi show how to use functional programming and algebraic data types to derive a pure functional definition of reactive programming like streams.
- **Pure JavaScript**: Christian Johansen shows you how you can up your game by leaving loops behind and embracing functions as the primary unit of abstraction.
- **Pure UI**: Guillermo Rauch discusses the definition of an application's UI as a pure function of application state.
- **PureScript**: PureScript is a strongly, statically typed language which compiles to JavaScript. It is written in and inspired by Haskell.
- **Ramda**: A practical library designed specifically for a functional programming style, one that makes it easy to create functional pipelines, one that never mutates user data.
  - **Practical Ramda - Functional Programming Examples**: Tom MacWright gives some practical examples of Ramda usage.
- **Ramda Fantasy**: Fantasy Land compatible types for easy integration with Ramda. This is an experimental project and will probably merge with Sanctuary.
- **Sanctuary**: Sanctuary is a functional programming library inspired by Haskell and PureScript. It depends on and works nicely with Ramda. Sanctuary makes it possible to write safe code without null checks.
  - **Sanctuary Build Script**: A build script for generating the Sanctuary website.
- **The Little Idea of Functional Programming**: Jack Hsu tries to take a look at a couple of simple concepts that make up the little idea behind functional programming and to tie the concepts back to code examples in JavaScript.
- **Timm**: Immutability helpers with fast reads and acceptable writes.
- **Transducers**: Transducers are a powerful and composable way to build algorithmic transformations that you can reuse in many contexts.
  - **"Transducers" Presentation at Strange Loop**: This talk will describe transducers, a new library feature for Clojure (but of interest to other languages) that emphasizes

composable, context-free, intermediate-free notions like 'mapping' and 'filtering' and their concrete reuse across all of the contexts above.

        - **Figuring out what transducers are good for**: Tim Cuthbertson attempts some plausible but detailed examples with Transducers in JavaScript.
        - **Implementations**: Libraries that implement Transducer protocoll and include ready to use transformers.
            - **Transduce**: Implementation by Kevin Beaty extracted from underarm.
            - **Transducers-js by Cognitect Labs**: A high performance Transducers implementation for JavaScript by Cognitect Labs.
            - **Transducers.js Library by James Long**: A small library for generalized transformation of data (inspired by Clojure's transducers)
                - **Transducers.js Round 2 with Benchmarks**: Refactored version of Transducers.js, some benchmarks, Laziness, the transformer protocoll.
                - **Transducers.js: A JavaScript Library for Transformation of Data**: A post announcing the transducers.js library with some explanation.
        - **Streaming Logs with Transducers and Ramda**: In this article we will use Ramda to parse a log file without curly braces (and introduce transducers along the way).
        - **Transducers Documentation for Clojure**: Transducers are composable algorithmic transformations. They are independent from the context of their input and output sources and specify only the essence of the transformation in terms of an individual element.
        - **Transducers Explained: Part 1**: An introduction to transducers using JavaScript. We will work from reducing over arrays, to defining transformations as transformers, then incrementally introducing transducers and using them with transduce.
        - **Transducers Explained: Pipelines**: In this article, we will introduce four new transducers: filter, remove, drop and take. We will show how transducers can be composed into pipelines and talk about the order of transformation.
        - **Transducers are Coming**: The first announcement by Rich Hickey.
        - **Transducers with Observable Sequences**: A chapter from the RxJS Book describing Transducers.
        - **Understanding Transducers in JavaScript**: Roman Liutikov translated code examples from similar Clojure article into JavaScript. So you can still read the article and check code examples here.
    - **Union Type**: Union types are a way to group different values together. Union-type is a small JavaScript library for defining and using union types.
- **Functional Reactive Programming (FRP)**: FRP is a programming paradigm for asynchronous dataflow programming using the building blocks of functional programming.
    - **A General Theory of Reactivity**: Kris Kowal describes popular primitives of Reactive Programming and some use cases.

- **A General Theory of Reactivity (Video)**: Kris Kowal talks about reactive primitives and their traits.
- **Controlling Time and Space**: This talk will quickly cover the basics of FRP, and then go into a couple different formulations of FRP that people are beginning to use. We will explore how these formulations fit together historically and theoretically.
- **Cycle.js**: A functional and reactive JavaScript framework that solves the cyclic dependency of Observables which emerge during dialogues (mutual observations) between the Human and the Computer.
  - **Async Driver**: Higher order factory for creating cycle.js async request based drivers. Allows you almost completely eliminate boilerplate code for this kind of drivers.
  - **Cycle.js Was Built to Solve Problems**: In this video André Staltz shows how Cycle.js has a practical purpose, meant to solve problems your customers/business may relate to.
  - **Cycle.js and Functional Reactive User Interfaces**: In this talk we will discover how Cycle.js is purely reactive and functional, and why it's an interesting alternative to React.
  - **Draw Cycle**: Simple Cycle.js program visualized
  - **Drivers**: Drivers are functions that listen to Observable sinks (their input), perform imperative side effects, and may return Observable sources (their output).
    - **Animation**: A Cycle driver for requestAnimationFrame.
    - **Audio Graph Driver**: Audio graph driver for Cycle.js based on virtual-audio-graph.
    - **Cookie**: Cycle.js Cookie Driver, based on cookie_js library.
    - **DOM**: The standard DOM Driver for Cycle.js based on virtual-dom, and other helpers.
    - **Fetch**: A Cycle.js Driver for making HTTP requests, using the Fetch API.
    - **Fetcher**: A Cycle.js Driver for making HTTP requests using stackable-fetcher.
    - **Firebase**: Thin layer around the firebase javascript API that allows you to query and declaratively update your favorite real-time database.
    - **HTTP**: A Cycle.js Driver for making HTTP requests, based on superagent.
    - **Hammer.js**: The driver incorporates the Hammer.js gesture library.
    - **History**: Cycle.js URL Driver based on the rackt/history library.
    - **Keys**: A Cycle.js driver for keyboard events.
    - **Mongoose.js**: A driver for using Mongoose with Cycle JS. Accepts both, write and read operations.
    - **Notification**: A Cycle.js Driver for showing and responding to HTML5 Notifications.
    - **Router**: A router built from the ground up with Cycle.js in mind. Stands on the shoulders of battle-tested libraries switch-path for route matching and rackt/history for dealing with the History API.
    - **Router5**: A source/sink router driver for Cycle.js, based on router5.

        - **Server-Sent Events**: Cycle.js driver for Server-Sent Events (SSE), a browser feature also known as EventSource. Server-Sent Events allow the server to continuously update the page with new events, without resorting to hacks like long-polling.
        - **Snabbdom**: Alternative DOM driver utilizing the snabbdom library.
        - **Socket.IO**: A Cycle driver for applications using Socket.IO
        - **Storage**: A Cycle.js Driver for using localStorage and sessionStorage in the browser.
    - **Example Projects**: Example applications built with Cycle.js
        - **Cycle.js Examples**: Browse and learn from examples of small Cycle.js apps using Core, DOM Driver, HTML Driver, HTTP Driver, JSONP Driver, and others.
        - **RX Marbles**: Interactive diagrams of Rx Observables.
        - **TODO: Minimum Viable Pizza**: Minimum Viable Pizza implemented with Cycle.js
        - **Tricycle**: A scratchpad for trying out Cycle.js.
    - **Intro to Functional Reactive Programming with Cycle.js**: Nick Johnstone gives an introduction to developing with Cycle.js in this video presentation.
    - **Learning How to Ride: an Introduction to Cycle.js**: In this talk, Fernando Macias Pereznieto introduces us to the good, the bad, and the beautiful of using Cycle.js, whether you are a complete beginner or an experienced JS ninja.
    - **Motorcycle.js**: This is a sister project that will continue to evolve and grow alongside Cycle.js for the foreseeable future. The primary focus of this project is to tune it for performance as much as possible.
        - **Most**: Monadic reactive streams with high performance.
    - **Plug and Play All Your Observable Streams With Cycle.js**: Frederik Krautwald explains the principles behind Cycle.js, it's inner workings and how to use it to create a simple program with drivers.
    - **Tricycle**: A scratchpad for trying out Cycle.js.
    - **What Developers Need to Know about MVI (Model-View-Intent)**: The article explains the general MVI pattern and how it relates to React, Reactive Programming and Cycle.js
- **Cycle.js and Functional Reactive User Interfaces**: In this talk we will discover how Cycle.js is purely reactive and functional, and why it's an interesting alternative to React.
- **Dynamics of Change: why Reactivity Matters**: In this talk we will see when passive or reactive strategy is advantageous, and how the reactive strategy is a sensible default.
- **Enemy of the State**: An introduction to Functional Reactive Programming and Bacon.js by Philip Roberts.
- **MobX**: MobX is a battle tested library that makes state management simple and scalable by transparently applying functional reactive programming.
- **Promises + FP = Beautiful Streams**: Yassine Elouafi show how to use functional programming and algebraic data types to derive a pure functional definition of reactive

programming like streams.

- **Stream Libraries**: Libraries which help you compose asynchronous operations on streams of time-varying values and events.
  - **Bacon.js**: A small functional reactive programming lib for JavaScript. Turns your event spaghetti into clean and declarative feng shui bacon, by switching from imperative to functional.
  - **Kefir.js**: Kefir — is a Reactive Programming library for JavaScript inspired by Bacon.js and RxJS, with focus on high performance and low memory usage.
  - **Most**: Monadic reactive streams with high performance.
  - **Reactive Extensions (RxJS)**: RxJS is a set of libraries for composing asynchronous and event-based programs using observable sequences and fluent query operators.
    - **Async JavaScript with Reactive Extensions**: Jafar Husain explains in this video how Netflix uses the Reactive Extensions (Rx) library to build responsive user experiences that strive to be event-driven, scalable and resilient.
    - **Exploring Rx Operators: FlatMap**: Christoph Burgdorf introduces the FlatMap operator and its usage for collections and observables.
    - **Exploring Rx Operators: Map**: Christoph Burgdorf explains how to use the map operator in RxJS.
    - **Functional Core Reactive Shell**: Giovanni Lodi makes an overview of different architecture meta-patterns and describes his current findings about functional programming and observables as a way to control side effects.
    - **Learn RX**: A series of interactive exercises for learning Microsoft's Reactive Extensions (Rx) Library for Javascript.
    - **Learn RxJS**: This site focuses on making RxJS concepts approachable, the examples clear and easy to explore, and features references throughout to the best RxJS related material on the web.
    - **Real World Observables**: Sergi Mansilla writes an FTP client to use it as an example for a real world application based on RxJS.
    - **Rx Training Games**: Rx Training Games is a coding playground that can be used to learn and practice Reactive Extensions coding grid-based games
    - **Rx-Book**: A complete book about RxJS v.4.0.
    - **RxMarbles**: A webapp for experimenting with diagrams of Rx Observables, for learning purposes.
    - **RxState**: Simple opinionated state management library based on RxJS and Immutable.js
    - **Taking Advantage of Observables in Angular 2**: Christoph Burgdorf describes the advantages of Observables and how you can use them in Angular 2 context.
    - **Transducers with Observable Sequences**: A chapter from the RxJS Book describing Transducers.

            - **Why We Built Xstream**: The authors needed a stream library tailored for Cycle.js. It needs to be “hot” only, small in kB size and it should have only a few and intuitive operators.
        - **Xstream**: An extremely intuitive, small, and fast functional reactive stream library for JavaScript.
            - **Why We Built Xstream**: The authors needed a stream library tailored for Cycle.js. It needs to be “hot” only, small in kB size and it should have only a few and intuitive operators.
    - **The Introduction to Reactive Programming**: André Staltz provides a complete introduction to the Reactive Programming and RxJS.
    - **What if the User was a Function?**: In this video André Staltz talks about the input/output cycle between humans and computers and how to take advantage of this model by using FRP and event streams.

Ability of a product to work with different input/output devices and rendering software. Including printers, email, mobile devices and different browsers.

- **Cross Browser**: Cross-browser refers to the ability of a website, web application, HTML construct or client-side script to function in environments that provide its required features and to bow out or degrade gracefully when features are absent or lacking.
    - **Can I use ... ?**: "Can I use" provides up-to-date browser support tables for support of front-end web technologies on desktop and mobile web browsers.
    - **Dev Tools by Microsoft**: These tools allow you to test your product on different version of Internet Explorer and Microsoft Edge.
    - **HTML5 Cross Browser Polyfills**: So here we're collecting all the shims, fallbacks, and polyfills in order to implant HTML5 functionality in browsers that don't natively support them.
    - **HTML5 Please**: Look up HTML5, CSS3, etc features, know if they are ready for use, and if so find out how you should use them – with polyfills, fallbacks or as they are.
    - **Modernizr**: It’s a collection of superfast tests – or “detects” as we like to call them – which run as your web page loads, then you can use the results to tailor the experience to the user.
    - **Normalize.css**: A modern, HTML5-ready alternative to CSS resets.
    - **Polyfill.io**: Just the polyfills you need for your site, tailored to each browser.
- **E-Mail**: Preparing HTML based electronic mail.
    - **Bulletproof E-Mail Buttons**: Design gorgeous buttons using progressively enhanced VML and CSS.
    - **Email Lab**: This a project for developing and testing email templates. It uses Grunt to streamline and simplify the creation of email templates. Email template can be built with re-usable components.

    - **Email-Boilerplate**: Use these code examples as a guideline for formatting your HTML email to avoid some of the major styling pitfalls in HTML email design.
    - **Foundation for Emails 2**: Frontend Framework for E-Mails including a grid, global styles, aligment classes, buttons, callout panels, thumbnail styles, typography, visibility classes.
    - **MJML**: MJML is a markup language designed to reduce the pain of coding a responsive email. Its semantic syntax makes it easy and straightforward and its rich standard components library speeds up your development time and lightens your email codebase.
    - **MailChimp E-Mail Blueprints**: Email Blueprints is a collection of HTML email templates that can serve as a solid foundation and starting point for the design of emails.
    - **Open Source Email Templates**: The sendwithus Open Source Template Project is a collection of free email templates created and managed by the sendwithus team and community.
    - **Really Simple Responsive HTML Email Template**: Sometimes all you want is a really simple HTML email template. Here it is.
    - **Responsive Email Design**: In this guide, the author will cover the fundamentals of designing and building a mobile-friendly email and back it all up with some neat tips and techniques.
    - **Responsive Email Templates**: Zurb Studios put together this set of super awesome email templates so that you can make your email campaigns responsive.
    - **The Ultimate Guide to CSS**: A complete breakdown of the CSS support for the top 10 most popular mobile, web and desktop email clients on the planet.
- **Keyboard**: Working with keyboard input in a web browser.
    - **What’s New with KeyboardEvents? Keys and Codes!**: Jeff Posnick talks about the code and key event attributes and how to use them in practice.
- **Mobile**: Development of websites optimized for viewing on smartphone and tablet devices.
    - **Emulation**: Tools for emulating features of mobile devices on a desktop.
        - **Responsinator**: Quickly test any website in popular resolutions.
        - **Simulate Mobile Devices with Chrome Developer Tools**: Use Chrome DevTools' Device Mode to build mobile-first, fully responsive web sites. Learn how to use it to simulate a wide range of devices and their capabilities.
        - **Touché**: Touché: bringing touch events to non-touch browsers (how touching!). No dependencies. No code bloat.
        - **thumbs.js**: Adds touch support to your browser.
    - **Gestures**: Resources for working with touch mechanics (what your fingers do on the screen) and touch activities (results of specific gestures).
        - **Hammer.js**: Hammer helps you add support for touch gestures to your page, and remove the 300ms delay from clicks.
        - **Introduction to Gestures**: Descriptions of different gestures an their meanings.
        - **Pointer Events Polyfill**: PEP polyfills pointer events in all browsers that haven't yet implemented them, providing a unified, responsive input model for all devices and input

types.
    - **Touchy**: Touchy is a jQuery plugin for managing touch events on W3C-compliant browsers, such as Mobile Safari or Android Browser, or any browser that supports the ontouchstart, ontouchmove and ontouchend events.
    - **jGestures**: A jQuery plugin that enables you to add gesture events just like native jQuery events. Includes event substitution for mouse events.
  - **Layout**: The way in which the parts of the website are arranged or laid out.
    - **Snap.js**: A Library for creating beautiful mobile shelfs (side menus) in Javascript.
    - **Swipe**: Swipe is the most accurate touch slider.
    - **Swiper**: Swiper is a free mobile touch slider with hardware accelerated transitions and native behavior. It is intended to be used in mobile websites, mobile web apps, and mobile native/hybrid apps.
    - **jqm-pagination**: A jQuery Mobile plugin for sequential pagination between pages with support for touch, mouse, and keyboard.
    - **swipeslide**: A Zepto Plugin for iOS like swipe navigation.
  - **Scrolling**: Native scrolling of the browsers doesn't always fit for mobile websites. There are resources which solve this problem.
    - **Overscroll**: Overscroll is a jQuery plug-in that mimics the iphone/ipad scrolling experience in a browser.
    - **Overthrow**: A framework-independent, overflow: auto polyfill for use in responsive design.
    - **Zynga Scroller**: A pure logic component for scrolling/zooming. It is independent of any specific kind of rendering or event system.
    - **iScroll**: iScroll is a high performance, small footprint, dependency free, multi-platform javascript scroller.
    - **jQuery.pep.js**: A lightweight plugin for kinetic-drag on mobile/desktop.
    - **jSwipeKinetic**: A jQuery plugin that enables you to add kinetic scrolling on your touch optimized projects. jSwipeKinetic is build on top of jGestures.
    - **pull-to-refresh.js**: This plugin enables a pull-to-refresh functionality in mobile safari for scrollable block elements with native scrolling on iOS.
  - **Tap Acceleration**: Every touch-based mobile browser has an artificial ~300ms delay between you tapping a thing on the screen and the browser considering it a "click", but there are ways to work around this behavior.
    - **300ms Tap Delay, Gone Away**: An article by Google describing the 300ms delay and how Chrome 32+ on Anrdoid deals with it.
    - **Hammer.js**: Hammer helps you add support for touch gestures to your page, and remove the 300ms delay from clicks.
    - **Tappable**: Tappable is a simple, standalone library to invoke the tap event for touch-friendly web browsers.

        - **fastclick**: FastClick is a simple, easy-to-use library for eliminating the 300ms delay between a physical tap and the firing of a click event on mobile browsers.
    - **Touch Keyboard**: Almost all modern smartphones provide a touch based keyboard for text input. There are some tactics to influence them and work around their quirks.
        - **A Guide To Designing Touch Keyboards**: In this article, we will look a bit deeper into the usability issues surrounding touch keyboards, including five design guidelines that will alleviate some of these pains.
    - **Working With Sensors**: All mobile devices are equipped with sensors like gyroscope, accelerometers, photometers, magnetometers and so on. Some of them are accessible in a browser through JavaScript.
        - **This End Up: Using Device Orientation**: In this article, we'll take a look at device orientation and motion events, and use CSS to rotate an image based on the orientation of the device.
        - **lenticular.js**: Tilt-controlled images in the browser.
- **Printers**: Manipulation of printer output through CSS.
    - **Tips And Tricks For Print Style Sheets**: A comprehensive guide for print optimization including background images and colors, expanding external links, QR codes, CSS3 filters for print quality.
- **Responsive Web Design (RWD)**: RWD responds to the needs of the users and the devices they're using. The layout changes based on the size and capabilities of the device.
    - **Data Tables**: Tables filled with data don't behave well on small screens. Here are some resources to tame them.
        - **Responsive Data Tables**: Several ideas by Chris Coyier on how to deal with responsive tables.
        - **stacktable.js**: jQuery plugin for stacking tables on small screens.
    - **Future Friendly Thinking**: We want to make things that are future friendly. The following ideas have been on our minds recently. Help us explore them further or suggest new ones.
    - **How to make a Responsive Newspaper-like layout**: The article describes several approaches for creating multi column websites.
    - **Images**: Images pose a set of problems on responsive websites: scaling, performance, retina screens and file size.
        - **Adaptive Images**: Adaptive Images detects your visitor's screen size and automatically creates, caches, and delivers device appropriate re-scaled versions of your web page's embeded HTML images.
        - **Choosing A Responsive Image Solution**: This article leads you through the basics, and then arms you with the information you'll need to pick the best responsive image solution for your situation.
        - **Clown Car Technique**: We can use media queries within SVG to serve up the right image. The beauty of the "Clown Car" technique is that all the logic remains in the SVG

file.
    - **How to Use Responsive Images...**: Engineers at Shutterstock describe different problems and solutions around responsive images.
    - **Picturefill**: A responsive image polyfill for , srcset, sizes, and more.
    - **Riloadr**: The goal of this library is to deliver optimized, contextual image sizes in responsive layouts that utilize dramatically different image sizes at different resolutions in order to improve page load time.
    - **Why We Need Responsive Images**: Tim Kadlec talks about page weight and responsive image solutions.
    - **imgLiquid**: A jQuery Plugin to resize images to fit in a container.
    - **jQuery Picture**: jQuery Picture is a tiny (2kb) plugin to add support for responsive images to your layouts. It supports both figure elements with some custom data attributes and the new proposed picture format.
  - **Monitoring Breakpoints**: Triggering JavaScript events on different breakpoints.
    - **Breakpoints.js**: Define breakpoints for your responsive design, and Breakpoints.js will fire custom events when the browser enters and/or exits that breakpoint.
    - **Harvey**: Harvey helps you monitor and manage behavior changes by firing an event whenever your media query is activated.
    - **enquire.js**: enquire.js is a lightweight, pure javascript library (with no dependencies) for programmatically responding to media queries.
  - **Navigation**: Adapting the website navigation to different screen sizes.
    - **Complex Navigation Patterns**: The article describes some emerging patterns for dealing with complex, lengthy and/or multi-level navigations.
    - **Responsive Navigation On Complex Websites**: To illustrate the techniques involved in implementing responsive navigation on a large website, author refers to two actual clients.
    - **Responsive Navigation Patterns**: The article describes some of the more popular techniques for handling navigation in responsive designs.
  - **Responsive Design Workflow**: In this video, Stephen Hay explores at a content-based approach to design workflow which is grounded in our multiplatform reality, not fixed-width Photoshop comps and overproduced wireframes.
  - **Responsive Elements**: Responsive elements makes it possible for any element to adapt and respond to the area they occupy. It's a tiny JavaScript library that you can drop into your projects today.
  - **Responsive Patterns**: A collection of patterns and modules for responsive designs.
  - **Text**: Working with text in a context of different viewport sizes.
    - **FitText**: FitText makes font-sizes flexible. Use this plugin on your fluid or responsive layout to achieve scalable headlines that fill the width of a parent element.
    - **Out Of Words!**: The responsive typography framework behind Words App.

        - **Responsive Font Sizing**: Making your font size respond to your screen size, easy & maintainable.
        - **Responsive Measure**: A jQuery plugin for generating a responsive ideal measure.
        - **Truly Fluid Typography With vh And vw Units**: This article describes viewport units and other technics to achieve typography which resizes smoothly with the screen.
    - **Viewport Component**: Viewport is a component to ease viewport management. You can get the dimensions of the viewport and beyond, which can be quite helpful to perform some checks with JavaScript.
- **Web Accessibility**: Web accessibility means that people with disabilities can perceive, understand, navigate, and interact with the Web, and that they can contribute to the Web.
    - **Notes on Using ARIA in HTML**: This document is a practical guide for developers on how to add accessibility information to HTML elements using the Accessible Rich Internet Applications specification.
    - **The A11Y Project**: A community-driven effort to make web accessibility easier.

Important developers, companies, organizations and news sources.

- **Communities Around Projects**: Successful open source projects attract many developers who produce plugins, libraries, tutorials and other resources. This section collects such resources.
    - **Angular**: AngularJS is a web application framework trying to address many of the challenges encountered in developing single-page applications.
        - **Adventures in Angular**: Adventures in Angular is a weekly podcast dedicated to the Angular JavaScript framework and related technologies, tools, languages, and practices.
        - **Angular 2 ESNext Starter**: This repo stands as a starting point for those who try Angular 2 in Javascript. It shows techniques how easy development can be also without Typescript.
        - **Angular 2 Template Syntax**: Victor Savkin writes about Angular 2 Templates including bindings, interpolation, syntax sugar, web component support and much more.
        - **Angular 2 Upgrade Strategies from Angular 1.x**: Some thoughts on general upgrading to Angular 2 and what you/your team can do to prepare.
        - **Building Redux in TypeScript with Angular 2**: In this post we’re going to discuss the ideas behind Redux. How to build our own mini version of the Redux Store and hook it up to Angular 2.
        - **Change Detection in Angular 2**: In this article Victor Savkin talks in depth about the Angular 2 change detection system.
        - **How to Implement Conditional Validation in Model-driven Forms**: In this article, we will learn about how to handle conditional validation in our model-driven form using the latest forms module.

    - **How to Prevent Name Collisions in Angular 2 Providers**: Opaque tokens are distinguishable and prevent us from running into naming collisions. Whenever we create a token that is not a type, OpaqueToken should be used.
    - **Ng-Newsletter**: The free, weekly newsletter of the best AngularJS content on the web.
    - **PrimeNG**: PrimeNG is a collection of rich UI components for AngularJS2. PrimeNG is a sibling of the popular JavaServer Faces Component Suite, PrimeFaces.
    - **Simple Language Translation**: Create a pipe that we can use to translate words in the HTML view and a service that we can use to translate our words in JS / Typescript.
    - **Using Model-Driven Forms with FormGroup and FormControl**: In this article, we will learn about building model-driven form with validation using the latest forms module, then we will talk about what are the advantages / disadvantages of using model driven form as compared to template-driven form.
  - **Backbone.js**: Backbone supplies structure to JavaScript-heavy applications by providing models, collections, views with declarative event handling, and connects it all to your existing application over a RESTful JSON interface.
  - **Bootstrap**: Bootstrap is a HTML, CSS, and JS framework for developing responsive, mobile first projects on the web.
    - **Bootstrap 4 Cheat Sheet**: A quick reference for Bootstrap v4 by Alexander Rechsteiner.
    - **Tree Shaking Bootstrap**: Jacob Parker describes how to include only those parts of Bootstrap you are really using on your website by leveraging CSS modules and ES6 modules.
  - **Cycle.js**: A functional and reactive JavaScript framework that solves the cyclic dependency of Observables which emerge during dialogues (mutual observations) between the Human and the Computer.
    - **Async Driver**: Higher order factory for creating cycle.js async request based drivers. Allows you almost completely eliminate boilerplate code for this kind of drivers.
    - **Cycle.js Was Built to Solve Problems**: In this video André Staltz shows how Cycle.js has a practical purpose, meant to solve problems your customers/business may relate to.
    - **Cycle.js and Functional Reactive User Interfaces**: In this talk we will discover how Cycle.js is purely reactive and functional, and why it's an interesting alternative to React.
    - **Draw Cycle**: Simple Cycle.js program visualized
    - **Drivers**: Drivers are functions that listen to Observable sinks (their input), perform imperative side effects, and may return Observable sources (their output).
      - **Animation**: A Cycle driver for requestAnimationFrame.
      - **Audio Graph Driver**: Audio graph driver for Cycle.js based on virtual-audio-graph.
      - **Cookie**: Cycle.js Cookie Driver, based on cookie_js library.
      - **DOM**: The standard DOM Driver for Cycle.js based on virtual-dom, and other helpers.

    - **Fetch**: A Cycle.js Driver for making HTTP requests, using the Fetch API.
    - **Fetcher**: A Cycle.js Driver for making HTTP requests using stackable-fetcher.
    - **Firebase**: Thin layer around the firebase javascript API that allows you to query and declaratively update your favorite real-time database.
    - **HTTP**: A Cycle.js Driver for making HTTP requests, based on superagent.
    - **Hammer.js**: The driver incorporates the Hammer.js gesture library.
    - **History**: Cycle.js URL Driver based on the rackt/history library.
    - **Keys**: A Cycle.js driver for keyboard events.
    - **Mongoose.js**: A driver for using Mongoose with Cycle JS. Accepts both, write and read operations.
    - **Notification**: A Cycle.js Driver for showing and responding to HTML5 Notifications.
    - **Router**: A router built from the ground up with Cycle.js in mind. Stands on the shoulders of battle-tested libraries switch-path for route matching and rackt/history for dealing with the History API.
    - **Router5**: A source/sink router driver for Cycle.js, based on router5.
    - **Server-Sent Events**: Cycle.js driver for Server-Sent Events (SSE), a browser feature also known as EventSource. Server-Sent Events allow the server to continuously update the page with new events, without resorting to hacks like long-polling.
    - **Snabbdom**: Alternative DOM driver utilizing the snabbdom library.
    - **Socket.IO**: A Cycle driver for applications using Socket.IO
    - **Storage**: A Cycle.js Driver for using localStorage and sessionStorage in the browser.
- **Example Projects**: Example applications built with Cycle.js
    - **Cycle.js Examples**: Browse and learn from examples of small Cycle.js apps using Core, DOM Driver, HTML Driver, HTTP Driver, JSONP Driver, and others.
    - **RX Marbles**: Interactive diagrams of Rx Observables.
    - **TODO: Minimum Viable Pizza**: Minimum Viable Pizza implemented with Cycle.js
    - **Tricycle**: A scratchpad for trying out Cycle.js.
- **Intro to Functional Reactive Programming with Cycle.js**: Nick Johnstone gives an introduction to developing with Cycle.js in this video presentation.
- **Learning How to Ride: an Introduction to Cycle.js**: In this talk, Fernando Macias Pereznieto introduces us to the good, the bad, and the beautiful of using Cycle.js, whether you are a complete beginner or an experienced JS ninja.
- **Motorcycle.js**: This is a sister project that will continue to evolve and grow alongside Cycle.js for the foreseeable future. The primary focus of this project is to tune it for performance as much as possible.
    - **Most**: Monadic reactive streams with high performance.

        - **Plug and Play All Your Observable Streams With Cycle.js**: Frederik Krautwald explains the principles behind Cycle.js, it's inner workings and how to use it to create a simple program with drivers.
        - **Tricycle**: A scratchpad for trying out Cycle.js.
        - **What Developers Need to Know about MVI (Model-View-Intent)**: The article explains the general MVI pattern and how it relates to React, Reactive Programming and Cycle.js
    - **Dojo Toolkit**: A JavaScript toolkit that saves you time and scales with your development process. Provides everything you need to build a Web app. Language utilities, UI components, and more, all in one place, designed to work together perfectly.
    - **Ember**: Ember.js is an open-source JavaScript web framework, based on the MVC pattern. It allows developers to create scalable single-page web applications.
        - **Bindings in Ember**: Unlike most other frameworks that include some sort of binding implementation, bindings in Ember.js can be used with any object.
        - **Router.js (Ember)**: Router.js is the routing microlib used by Ember.js.
    - **Foundation**: Foundation provides a responsive grid and HTML and CSS UI components, templates, and code snippets, including typography, forms, buttons, navigation and other interface components, as well as optional JavaScript extensions.
    - **Gulp**: Gulp is a toolkit that helps you automate painful or time-consuming tasks in your development workflow. It's very fast, platform-agnostic and simple.
        - **Articles & Tutorials**: Publications about gulp or step by step guides for setting up and using gulp in a project.
            - **Building with Gulp 3 and 4 (Series)**: Great series of articles about single components and gulp as a whole.
                - **Part 1: Examples**: Introduction to gulp and gulpfile.js.
                - **Part 2: Gulp's anatomy**: Orchestrator, Undertaker, Vinyl and Vinyl FS, Gulp Plugins.
                - **Part 3: Writing transformers**: Using map-stream, though2 and event-stream.
                - **Part 4: Incremental builds**: Building files which changed since last run and caching.
                - **Part 5: Caveats**: Error management in Gulp 3 and "MANY:1 disguised as a 1:1" problem.
            - **The vision, history, and future of the project (Apr. 2014)**: The article talks about Streams, Vinyl, Vinyl Adapters, Orchestrator and Error Management in Gulp 4.
            - **Why Gulp might not be the Answer**: ... there is still a conceptual problem that Gulp has yet to address. Many build steps are not 1:1 (one file in, one file out) but rather n:1 or 1:n.
        - **CSS**: Gulp plugins for working with CSS files.
            - **gulp-clean-css**: gulp plugin to minify CSS, using clean-css.
            - **gulp-cssnano**: Minify CSS with cssnano.

- **Concatenation**: Plugins for file concatenation. For example bundling CSS or JavaScript files.
  - **gulp-concat**: This plugin will concat files by your operating systems newLine. It will take the base directory from the first file that passes through it.
  - **gulp-group-concat**: Concats groups of files into a smaller number of files
- **Deployment**: Plugins for pushing built files into production.
  - **gulp-tar**: Create tarball from files.
  - **vinyl-ftp**: Blazing fast vinyl adapter for FTP.
  - **vinyl-s3**: Use S3 as a source or destination of vinyl files.
- **Ecosystem**: The network of developers and plugins around gulp.
  - **@sindresorhus plugins**: A collection of plugins by Sindre Sorhus.
  - **Gulp Friendly NPM Packages**: Normal node packages that work with gulp.
- **Filters**: Plugins for filtering files in a vinyl stream.
  - **gulp-cache**: A temp file based caching proxy task for gulp.
  - **gulp-cached**: A simple in-memory file cache for gulp.
  - **gulp-changed**: Only pass through changed files.
  - **gulp-filter**: Filter files in a vinyl stream.
  - **gulp-newer**: Pass through newer source files only.
  - **gulp-remember**: A plugin for gulp that remembers and recalls files passed through it.
  - **vinyl-diff**: This library allows you to perform diffs between streams of vinyl.
- **Images**: Plugins for working with images.
  - **gulp-imagemin**: Minify PNG, JPEG, GIF and SVG images.
  - **gulp-webp**: Convert PNG, JPEG, TIFF images to WebP.
- **JavaScript**: Module loaders, minifiers and other tools for working with JavaScript files.
  - **gulp-pure-cjs**: Gulp plugin for Pure CommonJS builder.
  - **gulp-uglify**: Minify files with UglifyJS.
  - **yoloader**: A CommonJS module loader implementation. It provides tools to bundle a CommonJS based project and to load such bundles.
- **SourceMaps**: A source map provides a way of mapping code within a compressed file back to it's original position in a source file.
  - **Plugins with gulp sourcemaps support**: A list of plugins which support gulp-sourcemaps.
  - **gulp-sourcemaps**: Source map support for Gulp.js
  - **vinyl-sourcemaps-apply**: Apply a source map to a vinyl file, merging it with preexisting source maps.
- **Utility**: Tools and parts for building gulp plugins.
  - **gulp-count**: Count files in a vinyl stream.

    - **gulp-debug**: Debug vinyl file streams to see what files are run through your gulp pipeline.
    - **gulp-size**: Logs out the total size of files in the stream and optionally the individual file-sizes.
    - **lazypipe**: Lazypipe allows you to create an immutable, lazily-initialized pipeline. It's designed to be used in an environment where you want to reuse partial pipelines, such as with gulp.
    - **map-stream**: Create a through stream from an asyncronous function.
  - **Vinyl**: Vinyl is a very simple metadata object that describes a file.
    - **gulp-chmod**: Change permissions of Vinyl files.
    - **gulp-rename**: A plugin to rename files easily.
    - **mem-fs**: Simple in-memory vinyl file store.
    - **vinyl-ast**: Parse-once and generate-once AST tool bridge for Gulp plugins.
    - **vinyl-buffer**: Creates a transform stream that takes vinyl files as input, and outputs buffered (isStream() === false) vinyl files as output.
    - **vinyl-file**: Create a vinyl file from an actual file.
    - **vinyl-fs**: Vinyl adapter for the file system.
    - **vinyl-fs-fake**: A vinyl adapter that extends vinyl-fs to allow for easy debugging by passing in virtual files instead of globs, and calling a function instead of writing.
    - **vinyl-git**: Vinyl adapter for git.
    - **vinyl-map**: Map vinyl files' contents as strings, so you can easily use existing code without needing yet another gulp plugin!
    - **vinyl-paths**: Get the file paths in a vinyl stream.
    - **vinyl-source-buffer**: Convert a text stream into a vinyl pipeline whose content is a buffer.
    - **vinyl-source-stream**: Use conventional text streams at the start of your gulp or vinyl pipelines, making for nicer interoperability with the existing npm stream.
    - **vinyl-to-stream**: Convert a vinyl stream to a text stream.
    - **vinyl-transform**: Wraps standard text transform streams so you can write fewer gulp plugins. Fulfills a similar use case to vinyl-map and vinyl-source-stream.
- **Meteor**: Meteor is a full-stack JavaScript platform for developing modern web and mobile applications. Meteor includes a key set of technologies for building connected-client reactive applications, a build tool, and a curated set of packages.
- **React**: React is a JavaScript library for creating user interfaces. Many people choose to think of React as the V in MVC. We built React to solve one problem: building large applications with data that changes over time.
  - **3 Lightweight React Alternatives**: Dan Prince explores Preact, VirtualDom & Deku.
  - **A Stateless React App?**: James K Nelson describes how to avoid state in React Components.

- **Block, Element, Modifying Your JavaScript Components**: Mark Dalgleish is discussing how to organize React code with BEM and build everything with Webpack.
- **CSS Modules To The Rescue.jsx**: If you use react-like templates/components, use webpack CSS loader to enable CSS Modules and forget about global CSS problems.
- **Find Your Perfect React Starter Project**: A simple search engine for React boilerplates with the ability to pick the ingredients.
- **Full-Stack Redux Tutorial**: We will go through all the steps of constructing a Node+Redux backend and a React+Redux frontend for a real-world application, using test-first development.
- **Functional DOM Programming**: One of the earliest intros to React and its purpose by Pete Hunt.
- **Functional Principles In React**: Jessica Kerr talks about four functional principles: Composition, Declarative Style, Isolation and Flow Of Data, and their usage in React.
- **Getting Started with TDD in React**: Learn how to test React components using a TDD approach with minimal setup, while learning exactly what to test and how to avoid common pitfalls.
- **Getting to Grips with React (as an Angular developer)**: In a series of posts Dave Ceddia tries to help you apply your hard-won knowledge of “Angularisms” to React.
- **How to Handle State in React. The Missing FAQ**: Osmel Mora challenges the common misconception that you always need a Flux-like architecture in your React apps.
- **How we use the Flux architecture in Delve**: Øystein Hallaråker describes how Delve utilizes the Flux application architecture.
- **Immutable Data and React**: Lee Byron talks about how persistent immutable data structures work, and techniques for using them in a React applications with Immutable.js.
- **JSX Transform**: JSX transpiler. A standard and configurable implementation of JSX decoupled from React.
- **Jest**: A JavaScript unit testing framework, used by Facebook to test services and React applications.
- **Model-View-Intent with React and RxJS**: Satish Chilukuri shows an example implementation of MVI pattern with React.
- **Monocle**: A developer tool for generating visual representations of your React app's component hierarchy.
- **Nothing New in React and Flux Except One Thing**: Andre Staltz talks about aspects of React and Flux which make them innovative and compelling.
- **Pure UI**: Guillermo Rauch discusses the definition of an application’s UI as a pure function of application state.
- **React - Basic Theoretical Concepts**: Sebastian Markbage attempts to formally explain his mental model of React. The intention is to describe this in terms of deductive reasoning that lead us to this design.

- **React App**: React App is a small library powered by React, Universal Router and History that handles routing, navigation and rendering logic in isomorphic (universal) and single-page applications.
- **React Components, Elements, and Instances**: Dan Abramov explains the Virtual DOM dictionary in React.
- **React Demystified**: This article is an attempt to explain the core ideas behind React.js and Virtual DOM.
- **React Native for Web**: This project allows components built upon React Native to be run on the Web, and it manages all component styling out-of-the-box.
- **React Starter Kit**: Isomorphic web app boilerplate including Node.js, Express, GraphQL, React.js, Babel 6, PostCSS, Webpack, Browsersync.
- **React Storybook**: Isolate your React UI Component development from the main app.
- **React Workshop**: This is a self-directed workshop. Follow along to the steps at your own pace, and feel free to ask your instructors questions as you go.
- **React in Patterns**: List of design patterns/techniques used while developing with React.
- **React vs Incremental DOM vs Glimmer**: In this post we will explore three technologies to build dynamic DOMs. We will also run benchmarks and find out which one is faster.
- **React: Rethinking best practices (2013)**: A video introduction to React by Pete Hunt.
- **ReactPerfTool**: ReactPerfTool tries to give you a more visual way of debugging performance of your React application. It does this by using the addons delivered by the React team and community to get measurements and visualize this using graphs.
- **Removing User Interface Complexity, or Why React is Awesome**: In this post James Long tries not to evangelize React specifically, but to explain why its technique is profound.
- **Rethinking Best Practices**: Pete Hunt talks about React's design decisions challenging established best practices.
- **Retractor**: Retractor exposes the internals of a React application for end-to-end testing purposes. This allows you to select DOM nodes based on the name of the React Component that rendered the node as well as its state or properties.
- **Some Problems with React/Redux**: André Staltz goes through the pros and cons of React + Redux.
- **Taming the React Setup**: Cody Lindley lays out seven React setups in this article and explains the relation of React to BYOA (Bring Your Own Architecture) approach.
- **Testing a React & Redux Codebase**: This series aims to be a very comprehensive guide through testing a React and Redux codebase, where you can really cover a lot with just unit tests because the code is mostly universal.
- **The Bare Minimum to Work with React**: Krasimir Tsonev describes how to start working with React after installing only 7 dependencies and learning only three commands.

        - **The Redux Ecosystem**: Let’s take a look at most of the features that you’ll have to deal with when the time comes,—and where React & Redux themselves can’t help you.
        - **The SoundCloud Client in React + Redux**: After finishing this step by step tutorial you will be able to author your own React + Redux project with Webpack and Babel.
        - **Tutorial: Cloning Yelp**: This post will guide you through building a full React app, even with little to no experience in the framework. We are going to build a Yelp clone in React.
        - **Using React to Sync Updates and Offline Activity**: Firas Durri describes how React based architectures make syncing state across devices much easier.
        - **What Developers Need to Know about MVI (Model-View-Intent)**: The article explains the general MVI pattern and how it relates to React, Reactive Programming and Cycle.js
        - **Why Did You Update?**: A function that monkey patches React and notifies you in the console when potentially unnecessary re-renders occur.
        - **Why did we build React?**: Pete Hunt tries to explain why Facebook devs built React in the first place.
    - **Yeoman**: Yeoman helps you to kickstart new projects, prescribing best practices and tools to help you stay productive. It provides a generator ecosystem.
    - **jQuery**: jQuery is a fast, small, and feature-rich JavaScript library. It makes things like HTML document traversal and manipulation, event handling, animation, and Ajax much simpler.
        - **Alternatives**: Other libraries which intend to replace jQuery in one way or another.
            - **Cash**: Cash is a small library for modern browsers that provides jQuery style syntax for manipulating the DOM.
            - **Chibi**: Chibi focuses on just the essentials, melted down and mixed with optimisation rainbows to create a really light micro-library that allows you to do awesome things.
            - **DOM CSS**: Small module for fast and reliable DOM styling.
            - **Minified.js**: Minified.js is a client-side JavaScript library that's both powerful and small. It offers jQuery-like features and utility functions with a single, consistent API.
            - **Plain.js**: Vanilla JS utilities for writing powerful web applications without jQuery.
            - **Zepto.js**: Zepto is a minimalist JavaScript library for modern browsers with a largely jQuery-compatible API.
        - **Authoring jQuery Plugins**: jQuery is an utility library and a plugin framework. This section collects resources about creating such plugins.
            - **Advanced Plugin Concepts**: A collection of best practices for jQuery plugin authoring.
            - **How to Create a Basic Plugin**: The article describes basic plugin creation and provides a simple boilerplate.
            - **Signs of a poorly written jQuery plugin**: Collection of jQuery plugin antipatterns.
            - **The Ultimate Guide to Writing jQuery Plugins**: A comprehensive guide on how to develop jQuery plugins including a simple boilerplate.

        - **Writing Stateful Plugins with the jQuery UI Widget Factory**: The article demonstrates the capabilities of the Widget Factory by building a simple progress bar plugin.
        - **jQuery Boilerplate**: This project won't seek to provide a perfect solution to every possible pattern, but will attempt to cover a simple template for beginners and above.
        - **jQuery Plugin Patterns**: This project won't seek to provide implementations for every possible pattern, but will attempt to cover popular patterns developers often use in the wild.
    - **Pragmatic jQuery Style**: Coding guidelines for working with jQuery.
    - **jQuery Fundamentals**: A guide to the basics of jQuery including a built-in editor for examples.
    - **jQuery UI**: jQuery UI is a curated set of user interface interactions, effects, widgets, and themes built on top of the jQuery JavaScript Library.
        - **Learning jQuery UI**: Series of articles about jQuery UI on learn.jquery.com.
- **News**: Websites & newsletters which provide daily and weekly news related to frontend web development.
    - **A Drip of JavaScript**: One quick JavaScript tip, delivered to your inbox every other week.
    - **CSS Weekly**: Weekly E-Mail roundup of CSS articles, tutorials, experiments and tools curated by Zoran Jambor.
    - **Deterministic**: A weekly digest of interesting news and articles covering functional programming for the web, especially on the front end.
    - **Frontend Dev Weekly**: Front-end developer news, tools and inspiration hand-picked each week.
    - **HTML5 Bookmarks**: Daily bookmarks of HTML5 related resources.
    - **HTML5 Weekly**: A once–weekly HTML5 and Web Platform technology roundup. CSS 3, Canvas, WebSockets, WebGL, Native Client, and more.
    - **JavaScript Weekly**: A free, once–weekly e-mail round-up of JavaScript news and articles.
    - **Ng-Newsletter**: The free, weekly newsletter of the best AngularJS content on the web.
    - **Responsive Design Newsletter**: A free, once–weekly round-up of responsive design articles, tools, tips, tutorials and inspirational links.
    - **WDRL**: A handcrafted, carefully selected list of web development related resources. Curated and published usually every week.
    - **Web Design Weekly**: A once a week email with no spam, no rambling. Just pure awesome links to the best news and articles to hit the interweb during the week.
    - **Web Platform Daily**: Daily digest of web development news.
    - **Web Tools Weekly**: Web Tools Weekly is a front-end development and web design newsletter with a focus on tools.

    - **echo.js**: Echo JS is a community-driven news site entirely focused on JavaScript development, HTML5, and front-end news.
- **Notable Community Members**: Important engineers, evangelists, architects and other celebrities.
    - **Addy Osmani**: Engineer at Google working on open web tooling.
        - **Learning JavaScript Design Patterns**: In this free book Addy Osmani explores applying both classical and modern design patterns to the JavaScript programming language.
        - **Patterns For Large-Scale JavaScript Application Architecture**: An extensive overview by Addy Osmani of existing architectural solutions in the frontend development field.
        - **Writing Modular JavaScript With AMD, CommonJS & ES Harmony**: In this article Addy Osmani reviewes several of the options available for writing modular JavaScript using modern module formats AMD, CommonJS and ES6 Modules.
    - **Alex Sexton**: Alex Sexton is an engineer at Stripe. He is on the Modernizr core team, the jQuery Board of Directors, as well as the Dojo Foundation Board.
        - **Modernizr**: It’s a collection of superfast tests – or “detects” as we like to call them – which run as your web page loads, then you can use the results to tailor the experience to the user.
    - **André Staltz**: Founder of the cycle.js framework and important contributor to ReactiveX.
        - **Cycle.js and Functional Reactive User Interfaces**: In this talk we will discover how Cycle.js is purely reactive and functional, and why it's an interesting alternative to React.
        - **Dynamics of Change: why Reactivity Matters**: In this talk we will see when passive or reactive strategy is advantageous, and how the reactive strategy is a sensible default.
        - **MVI in Cycle.js Docs**: André Staltz describes how to refactor an application into MVI pattern.
        - **Nothing New in React and Flux Except One Thing**: Andre Staltz talks about aspects of React and Flux which make them innovative and compelling.
        - **RxMarbles**: A webapp for experimenting with diagrams of Rx Observables, for learning purposes.
        - **Some Problems with React/Redux**: André Staltz goes through the pros and cons of React + Redux.
        - **The Introduction to Reactive Programming**: André Staltz provides a complete introduction to the Reactive Programming and RxJS.
        - **Unidirectional Data Flow Architectures (Talk)**: Andre Staltz compares modern architecture patterns including Flux, Redux, Model-View-Intent, Elm Arch and BEST.
        - **What if the User was a Function?**: In this video André Staltz talks about the input/output cycle between humans and computers and how to take advantage of this model by using FRP and event streams.

        - **Why We Built Xstream**: The authors needed a stream library tailored for Cycle.js. It needs to be “hot” only, small in kB size and it should have only a few and intuitive operators.
        - **Xstream**: An extremely intuitive, small, and fast functional reactive stream library for JavaScript.
            - **Why We Built Xstream**: The authors needed a stream library tailored for Cycle.js. It needs to be “hot” only, small in kB size and it should have only a few and intuitive operators.
    - **Brad Frost**: Web designer, speaker, writer, consultant, musician, and artist in beautiful Pittsburgh.
        - **Atomic Design**: Atomic Design discusses the importance of crafting robust design systems, and introduces a methodology for which to create smart, deliberate interface systems.
            - **A More Seamless Workflow—Style Guides for Better Design and Development**: Ash Connolly explains what styles guides are and which benefits they bring to designers and developers.
            - **Atomic Docs**: Atomic Docs is a styleguide generator and component manager. Atomic Docs is built in PHP. Inspired by Brad Frost's Atomic Design principles.
            - **Atomic Lab**: Template sharing and coding environment based on atomic design.
        - **Responsive Navigation Patterns**: The article describes some of the more popular techniques for handling navigation in responsive designs.
    - **Brian Lonsdorf**: Lead UXE Engineer at Salesforce, JavaScript developer and speaker known for his work in functional programming community.
        - **A Million Ways to Fold in JS**: Brian Lonsdorf provides many functional alternatives to loops in this video.
        - **Debugging Functional**: This post will demonstrate a simple solution that can go a long way to enhance the debugging experience in functional JavaScript applications.
        - **Free Monads Video Series**: A video series on free monads by Brian Lonsdorf explaining Coyoneda, Free Monad and Interpretors.
        - **Freeky**: Collection of free monads by Brian Lonsdorf.
        - **Hey Underscore, You're Doing It Wrong!**: In this talk Brian Lonsdorf gently takes a shot at underscore.js for not thinking about currying and partial function application in its library design.
        - **JSAir - Functional and Immutable Design Patterns in JavaScript**: An episode of JavaScript Air about "the how and why of functional programming and immutable design patterns in JavaScript" with Dab Abramov and Brian Lonsdorf as guests.
        - **Lenses Quick n’ Dirty**: A video by Brian Lonsdorf that introduces lenses.
        - **Lenses.js**: Composable kmett style lenses.
        - **Monad a Day: Reader**: Short video by Brian Lonsdorf about the Reader Monad.

    - **Monad a day 1: Reader**: A video by Brian Lonsdorf explaining the Reader Monad.
    - **Monad a day 2: Future**: Brian Lonsdorf explains the Future monad in this video.
    - **Monad a day 3: State**: Brian Lonsdorf explains the State monad in this video.
    - **Mostly Adequate Guide to Functional Programming**: A book by Brian Lonsdorf that introduces algebraic functional programming in JavaScript.
  - **Charles Max Wood**: Podcaster at Devchat.tv, organizer of remote confs such as Angular RC, React RC, Rails RC. Software consultant and developer.
    - **Adventures in Angular**: Adventures in Angular is a weekly podcast dedicated to the Angular JavaScript framework and related technologies, tools, languages, and practices.
    - **JavaScript Jabber**: A weekly podcast about JavaScript, including Node.js, Front-End Technologies, Careers, Teams and more.
  - **Chris Coyier**: Designer at Codepen. Writer at CSS-Tricks. Podcaster at ShopTalk.
    - **Responsive Data Tables**: Several ideas by Chris Coyier on how to deal with responsive tables.
  - **Douglas Crockford**: Computer programmer who is best known for his ongoing involvement in the development of the JavaScript language, for having popularized the data format JSON, and for developing JSLint and JSMin.
    - **Monads and Gonads**: In this video from YUIConf 2012, Douglas Crockford attempts to break the long-standing Monad tutorial curse by explaining the concept and applications of monads in a way that is actually understandable to the audience.
    - **Prototypal Inheritance in JavaScript**: An article by Douglas Crockford introducing the Object.create() method and describing the rational behind it.
  - **James Long**: Works on Firefox Developer Tools at Mozilla.
    - **A Study on Solving Callbacks with JavaScript Generators**: This article describes how Generators help fight callback hell.
    - **Removing User Interface Complexity, or Why React is Awesome**: In this post James Long tries not to evangelize React specifically, but to explain why its technique is profound.
    - **Transducers.js Library by James Long**: A small library for generalized transformation of data (inspired by Clojure's transducers)
      - **Transducers.js Round 2 with Benchmarks**: Refactored version of Transducers.js, some benchmarks, Laziness, the transformer protocoll.
      - **Transducers.js: A JavaScript Library for Transformation of Data**: A post announcing the transducers.js library with some explanation.
    - **Transducers.js Round 2 with Benchmarks**: Refactored version of Transducers.js, some benchmarks, Laziness, the transformer protocoll.
    - **Transducers.js: A JavaScript Library for Transformation of Data**: A post announcing the transducers.js library with some explanation.

    - **Jonathan Snook**: A web designer and developer who is currently UX Architect at Xero. Former lead frontend developer at Shopify.
        - **SMACSS**: SMACSS (pronounced “smacks”) is a way to examine your design process and as a way to fit those rigid frameworks into a flexible thought process. It is an attempt to document a consistent approach to site development when using CSS.
    - **Mikito Takada (mixu)**: Software engineer at Stripe.
        - **Learn CSS Layout the pedantic way**: Walks you through every major concept in CSS layout, without trying to simplify away the underlying mechanisms described in the CSS 2.1 and flexbox specs.
        - **Single Page Apps in Depth**: This free book is what I wanted when I started working with single page apps. It's not an API reference on a particular framework, rather, the focus is on discussing patterns, implementation choices and decent practices.
    - **Nicholas C. Zakas**: Former principal front-end engineer at Yahoo! and YUI developer. Leads a team of frontend engineers at Box now.
        - **Box Tech Talk: Scalable JavaScript Application Architecture**: A video by Nicholas Zakas (2012) about JavaScript Architecture.
        - **Scalable JavaScript Application Architecture**: In this video (2011) Nicholas Zakas discusses frontend architecture for complex, modular web applications with significant JavaScript elements.
        - **T3**: T3 is a minimalist JavaScript framework sponsored by Box Inc. that provides core structure to code.
        - **Understanding ECMAScript 6**: Free (as in pay what you want) E-Book by Nicholas C. Zakas describing the new features in EcmaScript 6.
    - **Nicolas Gallagher**: Frontend Engineer at Twitter.
        - **Normalize.css**: A modern, HTML5-ready alternative to CSS resets.
    - **Pete Hunt**: Co-founder & CEO @HelloSmyte. Ex-FB and Instagram. Worked on React.js.
        - **Functional DOM Programming**: One of the earliest intros to React and its purpose by Pete Hunt.
        - **React: Rethinking best practices (2013)**: A video introduction to React by Pete Hunt.
        - **Why did we build React?**: Pete Hunt tries to explain why Facebook devs built React in the first place.
- **Organizations**: Commercial companies and nonprofit organizations around web development.
    - **Airbnb**: Airbnb is a website for people to list, find, and rent lodging.
        - **Airbnb CSS + Sass Style Guide**: This style guide covers Terminology, Rule Declaration, Selectors, Properties, Formatting, Comments, OOCSS and BEM, ID Selectors, JavaScript hooks
          Sass, Syntax, Ordering, Mixins, Placeholders, Nested selectors.
        - **Airbnb JavaScript Style Guide**: A style guide for writing JavaScript code at Airbnb.

        - **Enzyme**: Enzyme is a JavaScript Testing utility for React that makes it easier to assert, manipulate, and traverse your React Components' output.
        - **Polyglot**: Polyglot.js is a I18n helper library written in JavaScript, made to work both in the browser and in CommonJS environments (Node). It provides a simple solution for interpolation and pluralization.
        - **Turbocharged JavaScript Refactoring with Codemods**: Joe Lencioni describes how they used codemods to transform a large JavaScript code base at AirBnB
    - **Box Inc.**: Box is an online file sharing and content management service for businesses based in Redwood City, California.
        - **Leche**: A JavaScript testing utility designed to work with Mocha and Sinon. This is intended for use both by Node.js and in browsers, so any changes must work in both locations.
        - **Nicholas C. Zakas**: Former principal front-end engineer at Yahoo! and YUI developer. Leads a team of frontend engineers at Box now.
            - **Box Tech Talk: Scalable JavaScript Application Architecture**: A video by Nicholas Zakas (2012) about JavaScript Architecture.
            - **Scalable JavaScript Application Architecture**: In this video (2011) Nicholas Zakas discusses frontend architecture for complex, modular web applications with significant JavaScript elements.
            - **T3**: T3 is a minimalist JavaScript framework sponsored by Box Inc. that provides core structure to code.
            - **Understanding ECMAScript 6**: Free (as in pay what you want) E-Book by Nicholas C. Zakas describing the new features in EcmaScript 6.
        - **Shalam**: A friendly tool for CSS spriting. Shalam allows you to add Retina-friendly, high-quality image sprites to your website without modifying any markup.
        - **T3**: T3 is a minimalist JavaScript framework sponsored by Box Inc. that provides core structure to code.
        - **stalker**: A jQuery plugin allowing elements to follow the user as they scroll a page.
    - **Facebook**: Facebook is a corporation and online social networking service headquartered in Menlo Park, California, in the United States.
        - **Immutable.js**: Immutable persistent data collections for Javascript which increase efficiency and simplicity.
        - **React**: React is a JavaScript library for creating user interfaces. Many people choose to think of React as the V in MVC. We built React to solve one problem: building large applications with data that changes over time.
            - **3 Lightweight React Alternatives**: Dan Prince explores Preact, VirtualDom & Deku.
            - **A Stateless React App?**: James K Nelson describes how to avoid state in React Components.

- **Block, Element, Modifying Your JavaScript Components**: Mark Dalgleish is discussing how to organize React code with BEM and build everything with Webpack.
- **CSS Modules To The Rescue.jsx**: If you use react-like templates/components, use webpack CSS loader to enable CSS Modules and forget about global CSS problems.
- **Find Your Perfect React Starter Project**: A simple search engine for React boilerplates with the ability to pick the ingredients.
- **Full-Stack Redux Tutorial**: We will go through all the steps of constructing a Node+Redux backend and a React+Redux frontend for a real-world application, using test-first development.
- **Functional DOM Programming**: One of the earliest intros to React and its purpose by Pete Hunt.
- **Functional Principles In React**: Jessica Kerr talks about four functional principles: Composition, Declarative Style, Isolation and Flow Of Data, and their usage in React.
- **Getting Started with TDD in React**: Learn how to test React components using a TDD approach with minimal setup, while learning exactly what to test and how to avoid common pitfalls.
- **Getting to Grips with React (as an Angular developer)**: In a series of posts Dave Ceddia tries to help you apply your hard-won knowledge of “Angularisms” to React.
- **How to Handle State in React. The Missing FAQ**: Osmel Mora challenges the common misconception that you always need a Flux-like architecture in your React apps.
- **How we use the Flux architecture in Delve**: Øystein Hallaråker describes how Delve utilizes the Flux application architecture.
- **Immutable Data and React**: Lee Byron talks about how persistent immutable data structures work, and techniques for using them in a React applications with Immutable.js.
- **JSX Transform**: JSX transpiler. A standard and configurable implementation of JSX decoupled from React.
- **Jest**: A JavaScript unit testing framework, used by Facebook to test services and React applications.
- **Model-View-Intent with React and RxJS**: Satish Chilukuri shows an example implementation of MVI pattern with React.
- **Monocle**: A developer tool for generating visual representations of your React app's component hierarchy.
- **Nothing New in React and Flux Except One Thing**: Andre Staltz talks about aspects of React and Flux which make them innovative and compelling.

- **Pure UI**: Guillermo Rauch discusses the definition of an application's UI as a pure function of application state.
- **React - Basic Theoretical Concepts**: Sebastian Markbage attempts to formally explain his mental model of React. The intention is to describe this in terms of deductive reasoning that lead us to this design.
- **React App**: React App is a small library powered by React, Universal Router and History that handles routing, navigation and rendering logic in isomorphic (universal) and single-page applications.
- **React Components, Elements, and Instances**: Dan Abramov explains the Virtual DOM dictionary in React.
- **React Demystified**: This article is an attempt to explain the core ideas behind React.js and Virtual DOM.
- **React Native for Web**: This project allows components built upon React Native to be run on the Web, and it manages all component styling out-of-the-box.
- **React Starter Kit**: Isomorphic web app boilerplate including Node.js, Express, GraphQL, React.js, Babel 6, PostCSS, Webpack, Browsersync.
- **React Storybook**: Isolate your React UI Component development from the main app.
- **React Workshop**: This is a self-directed workshop. Follow along to the steps at your own pace, and feel free to ask your instructors questions as you go.
- **React in Patterns**: List of design patterns/techniques used while developing with React.
- **React vs Incremental DOM vs Glimmer**: In this post we will explore three technologies to build dynamic DOMs. We will also run benchmarks and find out which one is faster.
- **React: Rethinking best practices (2013)**: A video introduction to React by Pete Hunt.
- **ReactPerfTool**: ReactPerfTool tries to give you a more visual way of debugging performance of your React application. It does this by using the addons delivered by the React team and community to get measurements and visualize this using graphs.
- **Removing User Interface Complexity, or Why React is Awesome**: In this post James Long tries not to evangelize React specifically, but to explain why its technique is profound.
- **Rethinking Best Practices**: Pete Hunt talks about React's design decisions challenging established best practices.
- **Retractor**: Retractor exposes the internals of a React application for end-to-end testing purposes. This allows you to select DOM nodes based on the name of the React Component that rendered the node as well as its state or properties.

        - **Some Problems with React/Redux**: André Staltz goes through the pros and cons of React + Redux.
        - **Taming the React Setup**: Cody Lindley lays out seven React setups in this article and explains the relation of React to BYOA (Bring Your Own Architecture) approach.
        - **Testing a React & Redux Codebase**: This series aims to be a very comprehensive guide through testing a React and Redux codebase, where you can really cover a lot with just unit tests because the code is mostly universal.
        - **The Bare Minimum to Work with React**: Krasimir Tsonev describes how to start working with React after installing only 7 dependencies and learning only three commands.
        - **The Redux Ecosystem**: Let's take a look at most of the features that you'll have to deal with when the time comes,—and where React & Redux themselves can't help you.
        - **The SoundCloud Client in React + Redux**: After finishing this step by step tutorial you will be able to author your own React + Redux project with Webpack and Babel.
        - **Tutorial: Cloning Yelp**: This post will guide you through building a full React app, even with little to no experience in the framework. We are going to build a Yelp clone in React.
        - **Using React to Sync Updates and Offline Activity**: Firas Durri describes how React based architectures make syncing state across devices much easier.
        - **What Developers Need to Know about MVI (Model-View-Intent)**: The article explains the general MVI pattern and how it relates to React, Reactive Programming and Cycle.js
        - **Why Did You Update?**: A function that monkey patches React and notifies you in the console when potentially unnecessary re-renders occur.
        - **Why did we build React?**: Pete Hunt tries to explain why Facebook devs built React in the first place.
    - **Regenerator**: This package implements a source transformation that takes the proposed syntax for generators/yield from future versions of JS and spits out efficient JS-of-today (ES5) that behaves the same way.
- **Google**: Google's mission is to organize the world's information and make it universally accessible and useful.
    - **Addy Osmani**: Engineer at Google working on open web tooling.
        - **Learning JavaScript Design Patterns**: In this free book Addy Osmani explores applying both classical and modern design patterns to the JavaScript programming language.
        - **Patterns For Large-Scale JavaScript Application Architecture**: An extensive overview by Addy Osmani of existing architectural solutions in the frontend development field.

        - **Writing Modular JavaScript With AMD, CommonJS & ES Harmony**: In this article Addy Osmani reviewes several of the options available for writing modular JavaScript using modern module formats AMD, CommonJS and ES6 Modules.
    - **Closure Compiler**: The Closure Compiler parses your JavaScript, analyzes it, removes dead code and rewrites and minimizes what's left. It also checks syntax, variable references, and types, and warns about common JavaScript pitfalls.
    - **Introducing Incremental DOM**: Incremental DOM is a library inspired by Virtual DOM developed at Google.
- **Microsoft**: Microsoft Corporation is an American multinational technology company, that develops, manufactures, licenses, supports and sells computer software, consumer electronics and personal computers and services.
    - **Dev Tools by Microsoft**: These tools allow you to test your product on different version of Internet Explorer and Microsoft Edge.
    - **Knockout.js**: Knockout is a standalone JavaScript implementation of the Model-View-ViewModel pattern with templates.
    - **Reactive Extensions (RxJS)**: RxJS is a set of libraries for composing asynchronous and event-based programs using observable sequences and fluent query operators.
        - **Async JavaScript with Reactive Extensions**: Jafar Husain explains in this video how Netflix uses the Reactive Extensions (Rx) library to build responsive user experiences that strive to be event-driven, scalable and resilient.
        - **Exploring Rx Operators: FlatMap**: Christoph Burgdorf introduces the FlatMap operator and its usage for collections and observables.
        - **Exploring Rx Operators: Map**: Christoph Burgdorf explains how to use the map operator in RxJS.
        - **Functional Core Reactive Shell**: Giovanni Lodi makes an overview of different architecture meta-patterns and describes his current findings about functional programming and observables as a way to control side effects.
        - **Learn RX**: A series of interactive exercises for learning Microsoft's Reactive Extensions (Rx) Library for Javascript.
        - **Learn RxJS**: This site focuses on making RxJS concepts approachable, the examples clear and easy to explore, and features references throughout to the best RxJS related material on the web.
        - **Real World Observables**: Sergi Mansilla writes an FTP client to use it as an example for a real world application based on RxJS.
        - **Rx Training Games**: Rx Training Games is a coding playground that can be used to learn and practice Reactive Extensions coding grid-based games
        - **Rx-Book**: A complete book about RxJS v.4.0.
        - **RxMarbles**: A webapp for experimenting with diagrams of Rx Observables, for learning purposes.

                - **RxState**: Simple opinionated state management library based on RxJS and Immutable.js
                - **Taking Advantage of Observables in Angular 2**: Christoph Burgdorf describes the advantages of Observables and how you can use them in Angular 2 context.
                - **Transducers with Observable Sequences**: A chapter from the RxJS Book describing Transducers.
                - **Why We Built Xstream**: The authors needed a stream library tailored for Cycle.js. It needs to be “hot” only, small in kB size and it should have only a few and intuitive operators.
        - **Visual Studio Code**: Build and debug modern web and cloud applications. VS Code is free and available on your favorite platform - Linux, Mac OSX, and Windows.
    - **Mozilla**: Mozilla is a community, which uses, develops, spreads and supports free software products. It is supported institutionally by the Mozilla Foundation and its tax-paying subsidiary, the Mozilla Corporation.
        - **Firefox**: Firefox is the highly popular free web browser. It is available for Linux, Mac, Windows, handheld devices, and in more than 70 different languages.
        - **James Long**: Works on Firefox Developer Tools at Mozilla.
            - **A Study on Solving Callbacks with JavaScript Generators**: This article describes how Generators help fight callback hell.
            - **Removing User Interface Complexity, or Why React is Awesome**: In this post James Long tries not to evangelize React specifically, but to explain why its technique is profound.
            - **Transducers.js Library by James Long**: A small library for generalized transformation of data (inspired by Clojure's transducers)
                - **Transducers.js Round 2 with Benchmarks**: Refactored version of Transducers.js, some benchmarks, Laziness, the transformer protocoll.
                - **Transducers.js: A JavaScript Library for Transformation of Data**: A post announcing the transducers.js library with some explanation.
            - **Transducers.js Round 2 with Benchmarks**: Refactored version of Transducers.js, some benchmarks, Laziness, the transformer protocoll.
            - **Transducers.js: A JavaScript Library for Transformation of Data**: A post announcing the transducers.js library with some explanation.
        - **Mozilla Developer Network (MDN)**: The MDN is a complete learning platform for Web technologies and the software that powers the Web.
        - **Nunjucks**: A rich and powerful templating language for JavaScript.
    - **Stripe**: Stripe is an Irish technology company that allows both private individuals and businesses to accept payments over the Internet.
        - **Alex Sexton**: Alex Sexton is an engineer at Stripe. He is on the Modernizr core team, the jQuery Board of Directors, as well as the Dojo Foundation Board.

        - **Modernizr**: It’s a collection of superfast tests – or “detects” as we like to call them – which run as your web page loads, then you can use the results to tailor the experience to the user.
    - **Mikito Takada (mixu)**: Software engineer at Stripe.
        - **Learn CSS Layout the pedantic way**: Walks you through every major concept in CSS layout, without trying to simplify away the underlying mechanisms described in the CSS 2.1 and flexbox specs.
        - **Single Page Apps in Depth**: This free book is what I wanted when I started working with single page apps. It's not an API reference on a particular framework, rather, the focus is on discussing patterns, implementation choices and decent practices.
    - **jquery.mobilePhoneNumber**: A general purpose library for validating and formatting mobile phone numbers.
    - **jquery.payment**: A general purpose library for building credit card forms, validating inputs and formatting numbers.
- **TODO Group**: TODO is an open group of companies who want to collaborate on practices, tools, and other ways to run successful and effective open source projects and programs.
- **Twitter**: Twitter is an online social networking service that enables users to send and read short 140-character messages called "tweets".
    - **Flight**: An event-driven web framework, from Twitter.
    - **Hogan.js**: Hogan.js is a 3.4k JS templating engine developed at Twitter. It was developed against the mustache test suite.
    - **Nicolas Gallagher**: Frontend Engineer at Twitter.
        - **Normalize.css**: A modern, HTML5-ready alternative to CSS resets.
- **World Wide Web Consortium (W3C)**: The W3C is an international community where Member organizations, a full-time staff, and the public work together to develop Web standards.
    - **Architecture of the World Wide Web: Identification**: This architecture document by W3C discusses the core design components of the Web. They are identification of resources, representation of resource state, and the protocols that support the interaction between agents and resources in the space.
    - **CSS Flexible Box Layout Module Level 1**: W3C specification for CSS flexbox.
    - **Document Object Model (DOM) Technical Reports**: Specifications by the W3C.
    - **Notes on Using ARIA in HTML**: This document is a practical guide for developers on how to add accessibility information to HTML elements using the Accessible Rich Internet Applications specification.
    - **Service Workers**: A method that enables applications to take advantage of persistent background processing, including hooks to enable bootstrapping of web applications while offline.

    - **Instant Loading Web Apps With An Application Shell Architecture**: Addy Osmani describes how to leverage the Service Worker API to drastically improve the loading speed of your web application.
    - **Introduction to Service Worker**: Matt Gaunt introduces the main features of Service Worker API in this article.
    - **Is ServiceWorker Ready?**: Tracks the implementation status across the main browsers.
  - **Shadow DOM W3C Editor's Draft**: This specification describes a method of combining multiple DOM trees into one hierarchy and how these trees interact with each other within a document, thus enabling better composition of the DOM.
- **Yandex**: Yandex is one of the largest internet companies in Europe, operating Russia’s most popular search engine and its most visited website.
  - **Block Element Modifier (BEM)**: Methodology aimed at achieving fast to develop long-lived projects, team scalability, and code reuse.
    - **A New Front-End Methodology: BEM**: An introduction by Varvara Stepanova at SmashingMagazine.
    - **An Introduction to the BEM Methodology**: General introduction article on tutsplus.
    - **BEM 101**: A collaborative post by Joe Richardson, Robin Rendle, and CSS-Tricks staff giving an introduction to BEM with some good examples.
    - **BEM I (finally) understand**: In this article Andrei Popa will focus on the basics of BEM and how to approach simple to complex anatomies.
    - **Battling BEM (Extended Edition): 10 Common Problems And How To Avoid Them**: This article aims to be useful for people who are already BEM enthusiasts and wish to use it more effectively or people who are curious to learn more about it.
    - **Block, Element, Modifying Your JavaScript Components**: Mark Dalgleish is discussing how to organize React code with BEM and build everything with Webpack.
    - **Emmet filter for BEM**: If you’re writing your HTML and CSS code in OOCSS-style, Yandex’s BEM style specifically, you will like this filter. It provides some aliases and automatic insertions of common block and element names in classes.
    - **Fifty Shades of BEM**: Article describes different flavors of BEM.
    - **How We Use BEM to Modularise Our CSS**: Andrei Popa describes the challenges, AlphaSights team had, implementing BEM in their projects.
    - **Introduction To BEM Methodology (Toptal)**: General introduction to BEM methodology and platform.
    - **MindBEMding – getting your head ’round BEM syntax**: Article on csswizardry explaining the BEM syntax for CSS classes.
    - **Pobem**: PostCSS plugin for BEM syntax.

            - **Support for BEM modules in Sass 3.3**: The next major release of Sass is poised for release and with it comes real support for BEM-style modules...
            - **To BEM or not to BEM**: A series of interviews on BEM methodology.
        - **Yandex Browser**: Chromium based browser developed by Yandex.
    - **Zurb**: ZURB is a product design company since 1998. Through consulting, product design tools and training, they transform the way businesses approach Progressive Design.
        - **Foundation**: Foundation provides a responsive grid and HTML and CSS UI components, templates, and code snippets, including typography, forms, buttons, navigation and other interface components, as well as optional JavaScript extensions.
        - **Foundation for Emails 2**: Frontend Framework for E-Mails including a grid, global styles, aligment classes, buttons, callout panels, thumbnail styles, typography, visibility classes.
        - **Responsive Email Templates**: Zurb Studios put together this set of super awesome email templates so that you can make your email campaigns responsive.
- **Podcasts**: A podcast is a form of digital media that consists of an episodic series of audio, video, digital radio, PDF, or ePub files subscribed to and downloaded automatically through web syndication or streamed online to a computer or mobile device.
    - **Adventures in Angular**: Adventures in Angular is a weekly podcast dedicated to the Angular JavaScript framework and related technologies, tools, languages, and practices.
    - **CDNify Podcasts**: The CDNify podcast covers all things tech, startup, web performance and acceleration.
    - **JavaScript Air**: The live broadcast podcast all about JavaScript.
    - **JavaScript Jabber**: A weekly podcast about JavaScript, including Node.js, Front-End Technologies, Careers, Teams and more.
    - **Non Breaking Space Show**: Seeking out the best, brightest, and smartest creative people on digital art, design, and development. From workflows to life hacks, we examine why they do what they do and how they did it.
    - **Shop Talk Show**: An internet radio show about the internet starring Dave Rupert and Chris Coyier.
    - **The Big Web Show**: The award winning Big Web Show features special guests and topics like web publishing, art direction, content strategy, typography, web technology, and more. It's everything web that matters.
    - **The Web Ahead**: Conversations with world experts on changing technologies and future of the web. The Web Ahead is your shortcut to keeping up.
    - **Web Security Warrior**: Web Security Warriors is a weekly discussion by developers about keeping websites, data, servers, and other internet outposts secure.

Programming/mark-up languages and web related standards.

- **Cascading Style Sheets (CSS)**: CSS are a stylesheet language used to describe the presentation of a document written in HTML or XML. It describes how elements should be rendered on screen, on paper, in speech, or on other media.
    - **CSS Coding Conventions**: Coding conventions are a set of guidelines for a specific programming language that recommend programming style, practices and methods for each aspect of a piece program written in this language.
        - **CSS Guidelines**: High-level advice and guidelines for writing sane, manageable, scalable CSS.
        - **Idiomatic CSS**: The following document outlines a reasonable style guide for CSS development. These guidelines strongly encourage the use of existing, common, sensible patterns.
        - **Maintainable CSS**: MaintainableCSS is an approach to writing modular, scalable and of course, maintainable CSS.
        - **Primer**: Primer is GitHub’s internal CSS framework. It includes basic global styling for typography, small components like buttons and tabs, and our general guidelines for writing HTML and CSS.
        - **Wordpress CSS Coding Standards**: The purpose of the WordPress CSS Coding Standards is to create a baseline for collaboration and review within various aspects of the WordPress open source project and community, from core code to themes to plugins.
    - **CSS Variables W3C Editor's Draft**: This module introduces cascading variables as a new primitive value type that is accepted by all CSS properties, and custom properties for defining them.
    - **Flexbox**: The Flexbox Layout officially called CSS Flexible Box Layout Module is new layout module in CSS3 made to improve the items align, directions and order in the container even when they are with dynamic or even unknown size.
        - **5 Flexbox Techniques You Need to Know About**: In this article we’re going to take a look at five flexbox approaches to solving common CSS layout problems. We’ve also included practical examples to showcase real life scenarios in which these techniques are applied.
        - **A Complete Guide to Flexbox**: Chris Coyer provides a great reference to the flexbox features with code examples.
        - **A Visual Guide to CSS3 Flexbox Properties**: Rather that explaining how the flex properties work, this guide will focus on how the flex properties affect the layout in a visual way.
        - **CSS Flexible Box Layout Module Level 1**: W3C specification for CSS flexbox.
        - **Flex-Grow is weird. Or is it?**: Manuel Matuzovic describes how flex-grow works, including it's weird quirks. Then he goes into several examples on how common layout patterns may be implemented using flex-grow and flex-basis.

        - **Flexbox Froggy**: A fun way to learn Flexbox by playing a game where you help Froggy and friends to arrive at a lilypad.
        - **Flexbox Patterns**: These interactive examples will show you practical ways to use Flexbox to build UI components.
        - **Flexbugs**: This repository is a community-curated list of flexbox issues and cross-browser workarounds for them.
    - **How To Center in CSS**: This tool consolidates the many ways of centering a div and gives you the code you need for each situation.
    - **The Complete Guide to Centering a DIV**: The aim of this article is to show how, with a few CSS tricks, any div can be centered; horizontally, vertically or both. And within the page or a div.
    - **Understanding border-image**: The new CSS3 property border-image can allow you to create flexible boxes with custom borders with a single div and a single image.
    - **What No One Told You About Z-Index**: The problem with z-index is that it's not complicated, but it if you've never taken the time to read its specification, there are almost certainly crucial aspects that you're completely unaware of.
- **Document Object Model (DOM)**: The DOM is a programming interface for HTML, XML and SVG documents. It defines methods that allow access to the tree, so that they can change the document structure, style and content.
    - **Document Events**: DOM event model is a generic event system and a set of standard modules of events for user interface control and document mutation notifications
        - **An Introduction To DOM Events**: Wilson Page introduces the basics of working with DOM events, then delves into their inner workings, explaining how we can make use of them to solve common problems.
        - **DOM Level 2 Event Model**: W3C specification section for DOM Level 2 Events.
        - **Gator**: Gator is a small (~0.8 kb minified + gzipped), simple, standalone, event delegation library.
    - **Overview**: High level guides, articles and documents about DOM.
        - **DOM Features You Didn't Know Existed**: Louis Lazaris talks about DOM Features you probably don't know.
        - **DOM Reference at the MDN**: Complete reference of the DOM provided by the Mozilla Development Network.
        - **Document Object Model (DOM) Technical Reports**: Specifications by the W3C.
        - **Introduction to the DOM**: This section provides a brief conceptual introduction to the DOM: what it is, how it provides structure for HTML and XML documents, how you can access it, and how this API presents the reference information and examples.
- **HyperText Markup Language (HTML)**: HTML is the standard markup language used to create web pages and its elements form the building blocks of all websites.

    - **Dive Into HTML5 (Book)**: Dive Into HTML5 elaborates on a hand-picked selection of features from the HTML5 specification and other fine standards.
    - **Google HTML/CSS Style Guide**: This document defines formatting and style rules for HTML and CSS. It aims at improving collaboration, code quality, and enabling supporting infrastructure.
    - **HEAD**: A list of everything that could go in the of your document.
    - **Idiomatic HTML**: The following document outlines a reasonable style guide for HTML development. These guidelines strongly encourage the use of existing, common, sensible patterns. They should be adapted as needed to create your own style guide.
    - **Video & Audio**: Use the HTML video and audio element to embed video content in a document.
        - **Bringing Production Video To The Web**: Stefan Lederer gives you a good overview of the state and future of video on the web.
    - **Wordpress HTML Coding Standards**: Coding conventions for WordPress.
- **Hypertext Transfer Protocol (HTTP)**: The Hypertext Transfer Protocol (HTTP) is an application protocol for distributed, collaborative, hypermedia information systems. HTTP is the foundation of data communication for the World Wide Web.
    - **Best Practice for HTTP2 Front-End Deployments**: This post describes how to set up and use the new HTTP/2 protocol.
    - **Ideal HTTP Performance**: This post talks about the design of the HTTP protocol, it's performance drawbacks and how to work around them.
- **JavaScript (EcmaScript)**: JavaScript is a full-fledged dynamic programming language that, when applied to an HTML document, can provide dynamic interactivity on websites. It is defined by ECMAScript standard.
    - **Control Flow & Error Handling**: Statements, that you can use to incorporate interactivity in your application.
        - **A Guide to Proper Error Handling in JavaScript**: Camilo Reyes describes ways to handle Exceptions and asynchronous errors in JavaScript.
    - **Enhancement Libraries**: Libraries that attempt to improve and enhance the vanilla JavaScript language by providing utility functions.
        - **Flow**: Flow is a static type checker for JavaScript. It can be used to catch common bugs in JavaScript programs before they run.
        - **Lodash**: A modern JavaScript utility library delivering modularity, performance, & extras.
        - **MOUT**: MOUT provides many helper methods similar to those found on other languages standard libraries (ie. Python, Ruby, PHP).
        - **Ramda**: A practical library designed specifically for a functional programming style, one that makes it easy to create functional pipelines, one that never mutates user data.
            - **Practical Ramda - Functional Programming Examples**: Tom MacWright gives some practical examples of Ramda usage.

        - **RubyJS**: RubyJS is a JavaScript implementation of all methods from Ruby classes like Array, String, Numbers, Time and more.
    - **Functions**: A function is a JavaScript procedure—a set of statements that performs a task or calculates a value.
        - **Closures explained by Jim Ley**: Explanation of closures in JavaScript.
        - **Let’s Learn JavaScript Closures**: So this post will be dedicated to the nuts and bolts of how and why closures work the way they do.
        - **MDN Guide Chapter about Functions**: Defining functions, scope, closures, arguments, parameters, arrow functions and predefined functions.
        - **MDN Reference for Functions**: Defining functions, arguments, parameters, methods, block-level functions and browser compatibility.
    - **Generators**: Generators allow you to define an iterative algorithm by writing a single function which can maintain its own state.
        - **A Closer Look at Generators Without Promises**: Author looks at libraries for asynchronous programming with generators but without promises.
        - **A Study on Solving Callbacks with JavaScript Generators**: This article describes how Generators help fight callback hell.
        - **Callbacks vs Coroutines**: In this post TJ Holowaychuk goes through hist experiences with coroutines and why he thinks they’re a great tool.
        - **Coroutine Event Loops in Javascript**: An intriguing use of coroutines is to implement event loops as an alternative to callback functions. This is particularly relevant to Javascript, where the use of callbacks is pervasive.
        - **Coroutine vs Continuation vs Generator**: StackOverflow Discussion about the difference between Couroutines, Continuations and Generators.
        - **Regenerator**: This package implements a source transformation that takes the proposed syntax for generators/yield from future versions of JS and spits out efficient JS-of-today (ES5) that behaves the same way.
    - **Grammar and Types**: JavaScript's basic grammar, variable declarations, data types scope, hoisting and literals.
        - **Detailed Overview of Well-Known Symbols**: Well-known symbols are used by built-in JavaScript algorithms. This article guides through the list of well-known symbols and explains how to use them comfortable in your code.
        - **Grammar and Types Chapter on the MDN**: This chapter discusses JavaScript's basic grammar, variable declarations, data types and literals.
        - **Variable Hoisting Explained**: The author explains how hoisting works in JavaScript including variable declarations and ES6 let operator.
        - **Variables Lifecycle: Why Let is not Hoisted**: ES2015 provides a different and improved mechanism for let. It demands stricter variable declaration practices and as result better code quality. Let's dive into more details about this process.

- **JS Coding Conventions**: Coding conventions are a set of guidelines for a specific programming language that recommend programming style, practices and methods for each aspect of a piece program written in this language.
  - **Airbnb JavaScript Style Guide**: A reasonable approach to JavaScript by Airbnb.
  - **Google JavaScript Style Guide**: JavaScript is the main client-side scripting language used by many of Google's open-source projects. This style guide is a list of dos and don'ts for JavaScript programs.
  - **Idiomatic.js**: The following list outlines the practices that Rick Waldron uses in all code that he is the original author of.
  - **JavaScript Standard Style**: A set of modules to check and improve the style of your code.
  - **WordPress JavaScript Coding Standards**: JavaScript has become a critical component in developing WordPress-based applications (themes and plugins) as well as WordPress core. Standards are needed for formatting and styling JavaScript code.
- **Objects**: An object is a software bundle of related state and behavior. Software objects are often used to model the real-world objects that you find in everyday life.
  - **ECMA-262-3 in detail: OOP - The general theory**: In this article we consider major aspects of object-oriented programming in ECMAScript. Much attention is given to theoretical aspects to see these processes from within.
  - **Gentle explanation of this keyword in JavaScript**: This article is focused on the invocation explanation, how the function call influences this and demonstrates the common pitfalls of identifying the context.
  - **OOP In JavaScript: What You NEED to Know**: In this article, we are concerned with only Inheritance and Encapsulation since only these two concepts apply to OOP in JavaScript.
  - **Object-Oriented Design in TypeScript / ES6**: The author dives into advanced OOP topics such as functional programming principles, inheritance and inversion of control.
  - **Prototypal Inheritance in JavaScript**: An article by Douglas Crockford introducing the Object.create() method and describing the rational behind it.
  - **Prototypal Object-Oriented Programming using JavaScript**: Mehdi Maujood describes the prototypical OO style and compares it to classes in JavaScript.
  - **Prototypes and Inheritance in JavaScript**: This article tries to demystify the concept of prototypes in JavaScript. It shows how prototypes allow objects to inherit functionality from other objects, an approach to building objects using the new operator and a constructor function.
- **Overview**: General, high level guides and introductions to the JavaScript language.
  - **Eloquent JavaScript (Book)**: A comprehensive book about JavaScript, the language, the browser and Node.js.

    - **JavaScript Garden**: JavaScript Garden is a growing collection of documentation about the most quirky parts of the JavaScript programming language. It gives advice to avoid common mistakes and subtle bugs.
    - **JavaScript Guide by Mozilla Developer Network**: The JavaScript Guide shows you how to use JavaScript and gives an overview of the language.
    - **Simplified JavaScript Jargon**: A community-driven attempt at explaining the loads of buzzwords making the current JavaScript ecosystem in a few simple words.
    - **Understanding ECMAScript 6**: Free (as in pay what you want) E-Book by Nicholas C. Zakas describing the new features in EcmaScript 6.
    - **What the heck is the event loop anyway?**: Philip Roberts, in this video, tries to create an intuitive understanding of what happens when JavaScript runs. He talks about the call stack, event loop, callback queue and other concepts.
    - **You Dont Know JS**: These books each take on specific core parts of the language which are most commonly misunderstood or under-understood, and dive very deep and exhaustively into them.
  - **Promises**: A promise represents the result of an asynchronous operation.
    - **Bluebird.js**: Bluebird is a full featured promise library with unmatched performance.
    - **Difference between a Promise and a Task**: Once you have a Promise instance the action has already started. Task instance does not run until someone calls .fork()
    - **ECMAScript Promises Spec**: Standard ES specification for promises.
    - **MDN page on Promises**: The Promise object is used for deferred and asynchronous computations. A Promise represents an operation that hasn't completed yet, but is expected in the future.
    - **The Promises/A+ Spec**: An open standard for sound, interoperable JavaScript promises—by implementers, for implementers.
    - **Tracking Unhandled Rejected Promises**: In Promise-based asynchronous code, rejections are used for error handling. One risk is that rejections may get lost, leading to silent failures.
    - **What is Promise.try, and why does it matter?**: In this brief article Sven Slootweg provides a better explanation of what Promise.try is, and why you should always use it, without exceptions.
    - **What's The Point Of Promises?**: The point of promises is to represent the eventual resulting value from an operation, but the reason to use them is to better parallel synchronous operations and to solve the callback hell.
- **JavaScript Object Notation (JSON)**: JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. It is based on a subset of the JavaScript Programming Language.
  - **JSON-LD**: JSON-LD is a lightweight Linked Data format. It is based on the already successful JSON format and provides a way to help JSON data interoperate at Web-scale.

- **Scalable Vector Graphics (SVG)**: An XML-based vector image format for two-dimensional graphics with support for interactivity and animation.
    - **Tools And Resources For Editing, Converting And Optimizing SVGs**: This article will provide you with tools and resources to simplify editing, converting, optimizing, and delivering SVGs.
- **Service Workers**: A method that enables applications to take advantage of persistent background processing, including hooks to enable bootstrapping of web applications while offline.
    - **Instant Loading Web Apps With An Application Shell Architecture**: Addy Osmani describes how to leverage the Service Worker API to drastically improve the loading speed of your web application.
    - **Introduction to Service Worker**: Matt Gaunt introduces the main features of Service Worker API in this article.
    - **Is ServiceWorker Ready?**: Tracks the implementation status across the main browsers.
- **Templating Languages and Engines**: Template engines are tools to separate program-logic and presentation into two independent parts. This makes the development of both logic and presentation easier, improves flexibility and eases modification and maintenance.
    - **Dot.js**: The fastest + concise javascript template engine for Node.js and browsers.
    - **Dust.js by LinkedIn**: Dust is a Javascript templating engine. It inherits its look from the ctemplate family of languages, and is designed to run asynchronously on both the server and the browser.
    - **Dōmo**: dōmo lets you write HTML markup and CSS styles in JavaScript syntax. It is a simpler and easier alternative to template engines and CSS pre-processors.
    - **HyperScript**: Create HyperText with JavaScript, on client or server.
    - **Marko**: Marko is a really fast and lightweight HTML-based templating engine from eBay. Marko runs on Node.js and in the browser and it supports streaming, async rendering and custom tags.
    - **Mustache**: Mustache is a Logic-less template language. There are no if statements, else clauses, or for loops. Instead there are only tags.
        - **Handlebars.js**: Handlebars.js is an extension to the Mustache templating language. Handlebars.js and Mustache are both logicless templating languages that keep the view and the code separated like we all know they should be.
        - **Hogan.js**: Hogan.js is a 3.4k JS templating engine developed at Twitter. It was developed against the mustache test suite.
        - **Mustache Specification**: This document explains the different types of Mustache tags.
        - **Walrus**: Walrus is a templating library inspired by mustache, handlebars, ejs and friends, but with a couple of important differences in philosophy and style.
        - **mustache.js**: mustache.js is an implementation of the mustache template system in JavaScript.

        - **templayed.js**: The fastest and smallest Mustache compliant Javascript templating library written in 1806 bytes.
    - **Nunjucks**: A rich and powerful templating language for JavaScript.
    - **Pithy**: An internal DSL for generating HTML in JavaScript.
    - **T**: T.js is a template engine that uses simple Javascript data structure to represent html/xml data.
    - **Template7**: Template7 is a mobile-first JavaScript template engine with Handlebars-like syntax. It is used as default template engine in Framework7.
    - **Transparency**: Transparency is a minimal template engine for jQuery. It maps JSON objects to DOM elements with zero configuration.
    - **Weld**: Weld binds data to markup, and can generate markup based on your data. There's no special syntax or data reshaping required.
- **Transpiled Languages**: Abstract languages converted to native, browser supported standards like JavaScript or CSS.
    - **ClojureScript**: ClojureScript is a compiler for Clojure that targets JavaScript. It is designed to emit JavaScript code which is compatible with the advanced compilation mode of the Google Closure optimizing compiler.
    - **Dart**: Dart is an open-source, scalable programming language, with robust libraries and runtimes, for building web, server, and mobile apps compiled to JavaScript
    - **Elm**: Elm is a functional programming language for declaratively creating web browser-based graphical user interfaces.
        - **The Elm Architecture**: The Elm Architecture is a simple pattern for infinitely nestable components. It is great for modularity, code reuse, and testing.
    - **Less**: Less is a CSS pre-processor, meaning that it extends the CSS language, adding features that allow variables, mixins, functions and many other techniques that allow you to make CSS that is more maintainable, themable and extendable.
    - **PureScript**: PureScript is a strongly, statically typed language which compiles to JavaScript. It is written in and inspired by Haskell.
    - **Sass**: Sass is an extension of CSS, adding nested rules, variables, mixins, selector inheritance, and more. It's translated to well-formatted, standard CSS using the command line tool or a web-framework plugin.
    - **Scala.js**: A Scala to JavaScript compiler.
    - **Stylus**: Stylus is a revolutionary new language, providing an efficient, dynamic, and expressive way to generate CSS. Supporting both an indented syntax and regular CSS style.
    - **TypeScript**: A typed superset of JavaScript that compiles to plain JavaScript. Popular in the Angular and Microsoft community.
        - **Angular 2: Why TypeScript?**: Angular 2 is written in TypeScript. In this article Victor Savkin talks about why they made the decision.

        - **InversifyJS**: A powerful and lightweight inversion of control container for JavaScript & Node.js apps powered by TypeScript.
        - **Safety in the Absence of Types**: Victor Savking talks about the limitation of TypeScript's static type checker and how to mitigate them.
- **Uniform Resource Identifier (URI)**: URI is a string of characters used to identify a resource. The most common form of URI is the Uniform Resource Locator (URL).
    - **Architecture of the World Wide Web: Identification**: This architecture document by W3C discusses the core design components of the Web. They are identification of resources, representation of resource state, and the protocols that support the interaction between agents and resources in the space.
    - **SpeakingURL**: This module aims to transliterate the input string and create a so-called Semantic or Speaking URL.
    - **URI.js**: URI.js is a javascript library for working with URLs. It offers a "jQuery-style" API to read and write all regular components and a number of convenience methods.
- **Web Animations API**: Web Animations is a new JavaScript API for driving animated content on the web. By unifying the animation features of SVG and CSS, Web Animations unlocks features previously only usable declaratively, and exposes powerful, high-performance animation capabilities to developers.
    - **Are we animated yet?**: This page tracks the progress of implementing the Web Animations API in Firefox.
    - **WAAPI Browser Support Test (+ Polyfill)**: This codepen tests whether and to which extend your browser supports Web Animations API. The test is run after including the Polyfill.
    - **Web Animations Polyfill**: JavaScript implementation of the Web Animations API.
- **WebAssembly**: WebAssembly is meant to fill a place that JavaScript has been forced to occupy up to now: a low-level code representation that can serve as a compiler target.
    - **7 Things You Should Know About WebAssembly**: In this post we will explore seven key facts about WebAssembly, one of the biggest changes the web will experience in the coming years.

Drop-in UI components for web sites and applications.

- **Buttons**: The term button refers to any graphical control element that provides the user a simple way to trigger an event, like searching for a query at a search engine, or to interact with dialog boxes, like confirming an action.
    - **Quantum Paper Buttons**: With this plugin you can hide any div behind a Quantum Paper Button or Qutton. Qunatum Paper is a digital paper that can change its size, shape and color to accommodate new content. Quantum paper is part of Google's new Material Design language.

    - **Sharingbuttons.io**: This generator outputs social media sharing buttons that do not use JavaScript, don't block your website from rendering, are accessible and don't track the user.
- **Code**: Code viewers and editors designed for embedding inside a website.
    - **Behave.js**: Behave.js is a lightweight library for adding IDE style behaviors to plain text areas, making it much more enjoyable to write code in.
    - **CodeMirror**: CodeMirror is a versatile text editor implemented in JavaScript for the browser. It is specialized for editing code, and comes with a number of language modes and addons that implement more advanced editing functionality.
    - **Intelligist**: A jQuery plugin that makes it easy to share and demo code in-page, using GitHub gists.
    - **Prism**: Prism is a lightweight, extensible syntax highlighter, built with modern web standards in mind.
    - **Rainbow**: Rainbow is a code syntax highlighting library written in Javascript.
      It was designed to be lightweight, easy to use, and extendable.
      It is completely themable via CSS.
    - **ansi_up**: A javascript library that converts text with ANSI terminal codes into colorful HTML
    - **tabIndent.js**: tabIndent.js enhances a textarea, so that the tab key no longer takes you to the next input, but rather, acts like a text editor by inserting a tab character.
- **Forms**: A HTML form on a web page allows a user to enter data that is sent to a server for processing. Web users fill out the forms using checkboxes, radio buttons, or text fields.
    - **ALAJAX**: A jQuery plugin to convert normal HTML forms into AJAX forms simply. It Ajaxifys your HTML Form with this plugin. No change will be required on Server-Side.
    - **Fields.js**: An abstract way of interacting with fields. Fields.js creates collections of fields. Each field is constantly evaluated for validity, and is accessible through the collection.
    - **Grid Forms**: A tiny Javascript/CSS framework that helps you make forms on grids with ease.
    - **HTML5Forms.js**: HTML5Forms.js is a JavaScript polyfill that implements a subset of the HTML5 Forms module in all browsers. The script will only add support for the different parts of the module when there doesn't exist a native implementation.
    - **Ladda**: Buttons with built-in loading indicators.
    - **Native form elements**: This is what every HTML5 form element looks like on your current operating system and browser.
    - **Redux Form**: A Higher Order Component using react-redux to keep form state in a Redux store.
    - **Serializers**: Libraries for collecting form data in JavaScript.
        - **form2js**: Convenient way to collect structured form data into JavaScript object.
        - **jQuery.serializeObject**: Encode a set of form elements as a JSON object for manipulation/submission.
        - **jquery-serialize-object**: Adds the method serializeObject to jQuery, to perform complex form serialization into JavaScript objects.

        - **jquery.serializeJSON**: Make an object out of form elements.
        - **serializeForm**: jQuery plugin to serialize form elements into an object.
    - **Validation**: A form validation behavior checks data against a set of criteria before passing it along to the server.
        - **Form Validation UX in HTML and CSS**: Chris Coyier describes how to implement form validation with just HTML attributes and some CSS trickery.
        - **Mailcheck.js**: The Javascript library and jQuery plugin that suggests a right domain when your users misspell it in an email address.
        - **One Validation**: This is a collection of regular expressions for general validation purposes. The basic design concept is to split up the regexes into semantic parts of the pattern to match.
        - **Parsley**: JavaScript form validation, without actually writing a single line of JavaScript!
    - **jQuery Super Labels Plugin**: This plugin was born out of the need to use the label-over-field method for forms.
- **Galeries & Image Sliders**: A sophisticated way to present a collection of images on your website.
    - **Lightgallery.js**: Full featured JavaScript Lightbox gallery without any dependencies.
- **Grid**: CSS Grid Layout Systems.
    - **Bourbon Neat**: A lightweight semantic grid framework for Sass and Bourbon.
    - **Profound Grid**: A responsive grid system for fixed and fluid layouts. Built in SCSS, it gives you flexibility and full control.
    - **RWDGrid**: 2kb, Mobile First Grid System, HTML5 Boilerplate Head, 960grid like naming convention. PSD Grid included.
    - **Simple Grid**: Simple Grid was created for developers who need a barebones grid. With fluid columns, Simple Grid is responsive down to mobile.
- **Rich Text Editors**: A rich text editor is the interface for editing rich text within web browsers. The aim is to reduce the effort for users trying to express their formatting directly as valid HTML markup.
    - **Content Sanitizers**: Rich text editors often produce unclean input when you copy & paste some content into them. Content sanitizers help you clean up the text.
        - **FilteredPaste.js**: A jQuery plugin that filters any pasted input so that your application gets clean input, without any tags or attributes that you don't want.
        - **Sanitize.js**: Sanitize.js is a whitelist-based HTML sanitizer. Given a list of acceptable elements and attributes, Sanitize.js will remove all unacceptable HTML from a DOM node.
        - **html-janitor**: Cleans up your markup and allows you to take control of your HTML. HTMLJanitor uses a defined whitelist to limit HTML it is given to a defined subset.
    - **Create.js**: Create.js is a comprehensive web editing interface for Content Management Systems. It is designed to provide a modern, fully browser-based HTML5 environment for managing content

- **Demarcate**: demarcate.js lets you edit directly in a page and generate Markdown back from the HTML elements.
- **Hallo**: Hallo is the simplest web editor imaginable. Instead of cluttered forms or toolbars, you edit your web content as it is. Just you, your web design, and your content.
- **Inspired by Medium**: Medium.com has a great and simple rich text editor built in. This libraries try to clone its behavior.
  - **Medium.js**: A tiny JavaScript library for making contenteditable beautiful (Like Medium's editor)
  - **Pen**: Rich text editor inspired by Medium and backed by Markdown.
  - **grande.js**: A small Javascript library that implements features from Medium's editing experience.
  - **medium-editor**: Medium.com WYSIWYG editor clone. Uses contenteditable API to implement a rich text solution.
- **Kajero**: Interactive JavaScript notebooks with markdown support and clever graphing.
- **MarkItUp**: markItUp! is a JavaScript plugin built on the jQuery library. It allows you to turn any textarea into a markup editor.
- **Mercury Editor**: Mercury is a full featured HTML5 editor. It was built from the ground up to help your team get the most out of content editing in modern browsers.
- **Quill**: Quill is a modern rich text editor built for compatibility and extensibility. It was created by Jason Chen and Byron Milligan and open sourced by Salesforce.com.
- **Scribe**: A rich text editor framework for the web platform, with patches for browser inconsistencies and sensible defaults. Developed by The Guardian.
  - **Inside the Guardian’s CMS: meet Scribe, an extensible rich text editor**: The team behind the Guardian’s digital content management system talk about how and why they built and open sourced Scribe.
- **Substance**: Substance is a JavaScript library for web-based content editing. It provides building blocks for realizing custom text editors and web-based publishing systems.
  - **Build your own editor with Substance**: This article describes the philosophy behind Substance and how to get started.
- **TextAngular**: A Lightweight, Two-Way-Bound Angular.js Text-Editor.
- **WYSIHTML5**: wysihtml5 is an open source rich text editor based on HTML5 technology and the progressive-enhancement approach. It aims to generate fully valid HTML5 markup by preventing unmaintainable tag soups and inline styles.
  - **Voog fork**: wysihtml is an extended and less strict approach on xing/wysihtml5 open source rich text editor based on HTML5 technology. The code is completely library agnostic: No jQuery, Prototype or similar is required.
  - **WYSIHTML5 Enhanced**: WYSIHTML5 Enhanced is a rich-text editor, based on the wonderful wysihtml5 editor, with a bit of help from Twitter Bootstrap, Font-Awesome, Jcrop and HTML5's Drag & Drop and File API.

        - **bootstrap3-wysiwyg**: Bootstrap-wysihtml5 is a javascript plugin that makes it easy to create simple, beautiful wysiwyg editors with the help of wysihtml5 and Twitter Bootstrap.
    - **X-editable**: This library allows you to create editable elements on your page. It can be used with any engine (bootstrap, jquery-ui, jquery only) and includes both popup and inline modes.
- **Table Of Contents**: Components for automatic table of contents generation.
    - **Tocbot**: Tocbot builds a table of contents (TOC) from headings in an HTML document.
- **UI Kits**: Collections of ready to use components.
    - **CloudFlare Components**: A set of UI components built by CloudFlare and based on React.
    - **Ink**: An HTML5/CSS3 framework used at SAPO for fast and efficient website design and prototyping.
    - **PrimeNG**: PrimeNG is a collection of rich UI components for AngularJS2. PrimeNG is a sibling of the popular JavaServer Faces Component Suite, PrimeFaces.
    - **Primer**: Primer is GitHub's internal CSS framework. It includes basic global styling for typography, small components like buttons and tabs, and our general guidelines for writing HTML and CSS.
    - **Pure.css**: A set of small, responsive CSS modules that you can use in every web project.
    - **UIkit**: A lightweight and modular front-end framework and a set of components for developing fast and powerful web interfaces.
    - **Vital**: A minimally invasive CSS framework for modern web applications.
- **Video & Audio**: Components for playing audio and video files on a website.
    - **Audio.js**: audio.js is a drop-in javascript library that allows HTML5's audio tag to be used anywhere.
    - **Howler.js**: howler.js is an audio library for the modern web. It defaults to Web Audio API and falls back to HTML5 Audio.
    - **JPlayer**: jPlayer a media library written in JavaScript. A jQuery plugin, jPlayer allows you to rapidly weave cross platform audio and video into your web pages.
    - **MediaElement.js**: HTML5 audio or video player with Flash and Silverlight shims that mimics the HTML5 MediaElement API, enabling a consistent UI in all browsers.
    - **Stratus 2**: Stratus is a jQuery powered SoundCloud player that lives at the bottom (or top) of your website or blog.
    - **Video.js**: Video.js is a web video player built from the ground up for an HTML5 world. It supports HTML5 and Flash video, as well as YouTube and Vimeo (through plugins). It supports video playback on desktops and mobile devices.

Task automation and asset delivery.

- **Automated Testing**: Automated software testing is a process in which software tools execute pre-scripted tests on a software application before it is released into production.

- **5 Common Misconceptions About TDD & Unit Tests**: Eric Elliott breaks down some common misconceptions and explains how you can benefit the most from TDD & unit tests.
- **A Gentle Introduction to Javascript Test Driven Development**: Over the course of the series, James Sinclair works through developing a full application in JavaScript that involves making network requests and manipulating the DOM.
- **Anti-patterns and Their Fixes**: Shane Tomlinson presents a sample application that contains several common anti-patterns and how these can be refactored to be more testable.
- **Chai**: Chai is a BDD/TDD assertion library for node and the browser that can be paired with any JavaScript testing framework.
- **Cucumber**: Cucumber is a software tool that computer programmers use for testing other software. It runs automated acceptance tests written in a behavior-driven development (BDD) style.
  - **Cucumber.js**: Cucumber.js is a Cucumber implementation written in pure JavaScript. It runs on Node.js, IO.js, browsers and any other JavaScript platform.
  - **Gherkin**: Gherkin is the language that Cucumber understands. It is a Business Readable, Domain Specific Language that lets you describe software's behaviour without detailing how that behaviour is implemented.
- **FiveUI**: FiveUI is an extensible tool for evaluating HTML user interfaces against sets of codified UI Guidelines.
- **Introducing BDD**: Dan North introduces behaviour-driven development (BDD). A software development process that emerged from test-driven development (TDD).
- **Jasmine**: Jasmine is a Behavior Driven Development testing framework for JavaScript. It does not rely on browsers, DOM, or any JavaScript framework. Thus it's suited for websites, Node.js projects, or anywhere that JavaScript can run.
- **JavaScript Testing: Unit vs Functional vs Integration Tests**: Unit tests, integration tests, and functional tests are all types of automated tests which form essential cornerstones of continuous delivery, a development methodology that allows you to safely ship changes to production in days or hours rather than months or years.
- **Jest**: A JavaScript unit testing framework, used by Facebook to test services and React applications.
- **Kakapo.js**: Kakapo its a full featured http mocking library, he allows you to entirely replicate your backend logic in simple and declaritive way directly in the browser.
- **Karma**: A simple tool that allows you to execute JavaScript code in multiple real browsers.
- **Leche**: A JavaScript testing utility designed to work with Mocha and Sinon. This is intended for use both by Node.js and in browsers, so any changes must work in both locations.
- **My Node Test Strategy**: Remy Sharp shates his automated testing process with tape, proxyquire, sinon and browserify.
- **PhantomCSS**: PhantomCSS takes screenshots and compares them to baseline images to test for RGB pixel differences. PhantomCSS then generates image diffs to help you find the

cause.

  - **QUnit**: QUnit is a powerful, easy-to-use JavaScript unit testing framework. It's used by the jQuery, jQuery UI and jQuery Mobile projects and is capable of testing any generic JavaScript code.
  - **Refactor Away Anti-Patterns**: Shane Tomlinson continues by refactoring the original application, including testing anti patterns, to be easier to read, easier to reuse, and easier to test.
  - **Sinon. JS Assertions for Chai**: Sinon–Chai provides a set of custom assertions for using the Sinon. JS spy, stub, and mocking framework with the Chai assertion library. You get all the benefits of Chai with all the powerful tools of Sinon. JS.
  - **Sinon.js**: Standalone test spies, stubs and mocks for JavaScript. No dependencies, works with any unit testing framework.
    - **How to Stub/Mock Complex Objects**: In this article, we'll look at how to stub objects which are deeply nested, and when functions have more complex return values and they interact with other objects.
  - **Tape**: Tap-producing test harness for node and browsers.
    - **Testing JavaScript Modules with Tape**: In this article we will get an in-depth look at three modules: tape, proxyquire, and sinon.
    - **Why I use Tape Instead of Mocha & So Should You**: Eric Elliott describes the advantages of Tape and compares it to more popular testing frameworks.
  - **TestCheck**: TestCheck is a library for generative testing of program properties, ala QuickCheck.
  - **Testing a React & Redux Codebase**: This series aims to be a very comprehensive guide through testing a React and Redux codebase, where you can really cover a lot with just unit tests because the code is mostly universal.
  - **Writing Testable JavaScript**: Rebecca Murphey discusses how to organize code to make JavaScript more testable in unit tests.
- **Build Tools**: Toolkits and their ecosystems, that help you automate painful and repeated tasks.
  - **Automaton**: Task automation tool built in JavaScript.
  - **Grunt**: Grunt is a task-based command line build tool for JavaScript projects.
    - **A beginner's guide to Grunt: Redux**: Simple Grunt boilerplate for frontend workflow with detailed instructions.
    - **GruntStart**: A Grunt-enabled head-start with the H5BP, jQuery, Modernizr, and Respond. The building blocks to quickly get started with Grunt to create an optimized website.
    - **Synchronised Testing Between Browsers/Devices**: The article describes an easy way to test your projects on your devices.
    - **Web development is getting complex. Let's go shopping.**: A step by step tutorial for building a new project with grunt.

- **Gulp**: Gulp is a toolkit that helps you automate painful or time-consuming tasks in your development workflow. It's very fast, platform-agnostic and simple.
  - **Articles & Tutorials**: Publications about gulp or step by step guides for setting up and using gulp in a project.
    - **Building with Gulp 3 and 4 (Series)**: Great series of articles about single components and gulp as a whole.
      - **Part 1: Examples**: Introduction to gulp and gulpfile.js.
      - **Part 2: Gulp's anatomy**: Orchestrator, Undertaker, Vinyl and Vinyl FS, Gulp Plugins.
      - **Part 3: Writing transformers**: Using map-stream, though2 and event-stream.
      - **Part 4: Incremental builds**: Building files which changed since last run and caching.
      - **Part 5: Caveats**: Error management in Gulp 3 and "MANY:1 disguised as a 1:1" problem.
    - **The vision, history, and future of the project (Apr. 2014)**: The article talks about Streams, Vinyl, Vinyl Adapters, Orchestrator and Error Management in Gulp 4.
    - **Why Gulp might not be the Answer**: ... there is still a conceptual problem that Gulp has yet to address. Many build steps are not 1:1 (one file in, one file out) but rather n:1 or 1:n.
  - **CSS**: Gulp plugins for working with CSS files.
    - **gulp-clean-css**: gulp plugin to minify CSS, using clean-css.
    - **gulp-cssnano**: Minify CSS with cssnano.
  - **Concatenation**: Plugins for file concatenation. For example bundling CSS or JavaScript files.
    - **gulp-concat**: This plugin will concat files by your operating systems newLine. It will take the base directory from the first file that passes through it.
    - **gulp-group-concat**: Concats groups of files into a smaller number of files
  - **Deployment**: Plugins for pushing built files into production.
    - **gulp-tar**: Create tarball from files.
    - **vinyl-ftp**: Blazing fast vinyl adapter for FTP.
    - **vinyl-s3**: Use S3 as a source or destination of vinyl files.
  - **Ecosystem**: The network of developers and plugins around gulp.
    - **@sindresorhus plugins**: A collection of plugins by Sindre Sorhus.
    - **Gulp Friendly NPM Packages**: Normal node packages that work with gulp.
  - **Filters**: Plugins for filtering files in a vinyl stream.
    - **gulp-cache**: A temp file based caching proxy task for gulp.
    - **gulp-cached**: A simple in-memory file cache for gulp.
    - **gulp-changed**: Only pass through changed files.
    - **gulp-filter**: Filter files in a vinyl stream.

    - **gulp-newer**: Pass through newer source files only.
    - **gulp-remember**: A plugin for gulp that remembers and recalls files passed through it.
    - **vinyl-diff**: This library allows you to perform diffs between streams of vinyl.
- **Images**: Plugins for working with images.
    - **gulp-imagemin**: Minify PNG, JPEG, GIF and SVG images.
    - **gulp-webp**: Convert PNG, JPEG, TIFF images to WebP.
- **JavaScript**: Module loaders, minifiers and other tools for working with JavaScript files.
    - **gulp-pure-cjs**: Gulp plugin for Pure CommonJS builder.
    - **gulp-uglify**: Minify files with UglifyJS.
    - **yoloader**: A CommonJS module loader implementation. It provides tools to bundle a CommonJS based project and to load such bundles.
- **SourceMaps**: A source map provides a way of mapping code within a compressed file back to it’s original position in a source file.
    - **Plugins with gulp sourcemaps support**: A list of plugins which support gulp-sourcemaps.
    - **gulp-sourcemaps**: Source map support for Gulp.js
    - **vinyl-sourcemaps-apply**: Apply a source map to a vinyl file, merging it with preexisting source maps.
- **Utility**: Tools and parts for building gulp plugins.
    - **gulp-count**: Count files in a vinyl stream.
    - **gulp-debug**: Debug vinyl file streams to see what files are run through your gulp pipeline.
    - **gulp-size**: Logs out the total size of files in the stream and optionally the individual file-sizes.
    - **lazypipe**: Lazypipe allows you to create an immutable, lazily-initialized pipeline. It's designed to be used in an environment where you want to reuse partial pipelines, such as with gulp.
    - **map-stream**: Create a through stream from an asyncronous function.
- **Vinyl**: Vinyl is a very simple metadata object that describes a file.
    - **gulp-chmod**: Change permissions of Vinyl files.
    - **gulp-rename**: A plugin to rename files easily.
    - **mem-fs**: Simple in-memory vinyl file store.
    - **vinyl-ast**: Parse-once and generate-once AST tool bridge for Gulp plugins.
    - **vinyl-buffer**: Creates a transform stream that takes vinyl files as input, and outputs buffered (isStream() === false) vinyl files as output.
    - **vinyl-file**: Create a vinyl file from an actual file.
    - **vinyl-fs**: Vinyl adapter for the file system.

        - **vinyl-fs-fake**: A vinyl adapter that extends vinyl-fs to allow for easy debugging by passing in virtual files instead of globs, and calling a function instead of writing.
        - **vinyl-git**: Vinyl adapter for git.
        - **vinyl-map**: Map vinyl files' contents as strings, so you can easily use existing code without needing yet another gulp plugin!
        - **vinyl-paths**: Get the file paths in a vinyl stream.
        - **vinyl-source-buffer**: Convert a text stream into a vinyl pipeline whose content is a buffer.
        - **vinyl-source-stream**: Use conventional text streams at the start of your gulp or vinyl pipelines, making for nicer interoperability with the existing npm stream.
        - **vinyl-to-stream**: Convert a vinyl stream to a text stream.
        - **vinyl-transform**: Wraps standard text transform streams so you can write fewer gulp plugins. Fulfills a similar use case to vinyl-map and vinyl-source-stream.
    - **Mimosa**: Mimosa is a batteries included web development workflow tool that will get you coding in seconds rather than hunting down plugins and wrangling config for hours.
    - **Plop**: Micro-generator framework that makes it easy for an entire team to create files with a level or uniformity.
        - **Automating Workflow with plop**: Automating UI Development with Riot.js and ES6 Javascript.
    - **Webpack**: Webpack is a module bundler. It takes modules with dependencies and generates static assets representing those modules.
        - **Block, Element, Modifying Your JavaScript Components**: Mark Dalgleish is discussing how to organize React code with BEM and build everything with Webpack.
        - **Developing with Docker and Webpack**: Chris Harrington explains how to create a development environment with Webpack and Docker to match the production as much as possible.
        - **Full-Stack Redux Tutorial**: We will go through all the steps of constructing a Node+Redux backend and a React+Redux frontend for a real-world application, using test-first development.
        - **How to Set Up Webpack Image Loader**: This brief tutorial will help you set up an image loader in Webpack.
        - **The SoundCloud Client in React + Redux**: After finishing this step by step tutorial you will be able to author your own React + Redux project with Webpack and Babel.
        - **Webpack from Apprentice to Master**: The purpose of this guide is to help you get started with Webpack and then go beyond basics.
        - **WebpackBin**: A webpack code sandbox.
        - **Why I think Webpack is the Right Approach To Build Pipelines**: Thomas Boyt compares how Grunt, Gulp, Broccoli and Webpack discover dependencies.

  - **Yeoman**: Yeoman helps you to kickstart new projects, prescribing best practices and tools to help you stay productive. It provides a generator ecosystem.
- **CSS Tools**: Tools for analysis, pre and post processing of CSS files.
  - **CSS Pack**: Packs CSS dependency graphs produced from dgraph or module-deps into a single CSS bundle, assuming every node in the graph contains CSS source and the graph itself is sorted with deps-sort
  - **CSS Stringify**: CSS stringifier using the AST from 'css.parse'
  - **CSSCSS**: A CSS redundancy analyzer that analyzes redundancy.
  - **Clean CSS**: Clean-css is a fast and efficient Node.js library for minifying CSS files.
  - **Helium CSS**: Helium is a tool for discovering unused CSS across many pages on a web site.
  - **PostCSS**: PostCSS parses CSS into an abstract syntax tree (AST), passes it through a series of plugins, and then concatenates back into a string.
    - **An Introduction to PostCSS**: This article describes what PostCSS is and how to get started.
    - **ES CSS Modules**: PostCSS plugin that combines CSS Modules and ES Imports.
    - **Improving the Quality of Your CSS with PostCSS**: In this article, we will explore how we can utilise PostCSS to help us maintain a higher quality in our CSS code.
    - **React Starter Kit**: Isomorphic web app boilerplate including Node.js, Express, GraphQL, React.js, Babel 6, PostCSS, Webpack, Browsersync.
    - **Working with Images in Stylesheets**: Aleks Hudochenkov does a great job of showcasing what PostCSS is good at and the role it has grown into in the front end stack.
  - **Stylelint**: Stylelint's ambitious goal is to supplement our discipline with automatic enforcement — to provide a core set of rules and a pluggable framework that CSS authors can use to enforce their own strategies.
    - **Lint your CSS with Stylelint**: David Clark writes about reasons for using a CSS linter and advantages of Stylelint.
- **Code Editors**: Text editor programs designed specifically for editing source code of a website.
  - **Atom**: Atom is a text editor that's modern, approachable, yet hackable to the core—a tool you can customize to do anything but also use productively without ever touching a config file.
  - **Brackets**: An open source code editor for the web, written in JavaScript, HTML and CSS.
  - **Notepad++**: Notepad++ is a free (as in "free speech" and also as in "free beer") source code editor and Notepad replacement that supports several languages. Running in the MS Windows environment, its use is governed by GPL License
  - **Visual Studio Code**: Build and debug modern web and cloud applications. VS Code is free and available on your favorite platform - Linux, Mac OSX, and Windows.
- **Documentation**: Writing, generating, publishing and consuming documentation for web deliverables.

  - **Atomic Docs**: Atomic Docs is a styleguide generator and component manager. Atomic Docs is built in PHP. Inspired by Brad Frost's Atomic Design principles.
  - **Daux**: Daux.io is a documentation generator that uses a simple folder structure and Markdown files to create custom documentation on the fly.
  - **Dexy**: Dexy makes it easier to create technical documents by doing the repetitive parts for you. Dexy provides a consistent interface to tools and scripts so you don't have to run them manually.
  - **Docco**: Docco is a quick-and-dirty documentation generator. It produces an HTML document that displays your comments intermingled with your code.
  - **JSDoc Documentation**: Comprehensive guide for JSDoc.
  - **Ronn**: Ronn builds manuals. It converts simple, human readable textfiles to roff for terminal display, and also to HTML for the web.
  - **Transcribe**: Transcribe is a simple program which generates Markdown documentation from code comments.
  - **YUIDoc**: YUIDoc is a Node.js application that generates API documentation from comments in source, using a syntax similar to tools like Javadoc and Doxygen.
  - **coddoc**: coddoc is a jsdoc parsing library. Coddoc is different in that it is easily extensible by allowing users to add tag and code parsers. It also parses source code to be used in APIs.
  - **devdocs.io**: Devdocs is an all-in-one API documentation reader with a fast, organized, and consistent interface.
  - **dox**: JavaScript documentation generator for node using markdown and jsdoc.
  - **styledocco**: StyleDocco generates documentation and style guide documents from your stylesheets.
- **Fonts for Programmers**: Programmers need special fonts, which help align the code and distinguish between characters, that look alike.
  - **Droid Sans Mono**: Droid Sans Mono makes for a great programming font. Its only real flaw is the lack of a slashed zero.
  - **Free Programming Fonts**: A demonstration of beautiful fonts for people who love to code.
  - **Mononoki**: Mononoki is a typeface by Matthias Tellen, created to enhance code formatting.
  - **Profont**: Profont is a monospaced font created to be a most readable font for programming. It is designed to look good a really small sizes
  - **Source Code Pro**: Source Code Pro is a set of OpenType fonts that have been designed to work well in user interface (UI) environments.
  - **Space Mono**: Space Mono is an original fixed-width type family designed by Colophon Foundry for Google Design.
- **Getting Started**: Step by step guides for setting up a frontend development workflow.
  - **Front-end Process - Flat Builds and Automation (series)**: A flat build is basically the process of coding a static page (or pages) in HTML and CSS. The idea is to supply our

developers with design assets such as style guides, pattern libraries or prototypes, including assets such as images, fonts, css, and javascript, as flat builds.
    - **CSS Framework (Inuit)**: In this part the author introduces the inuit CSS framework and describes how to integrate the framework into the development process.
    - **Environment Setup & Yeoman**: In this part the author outlines how to set up your dev environment, and how to scaffold a project using Yeoman.
    - **Grunt Tasks**: The author outlined how to set up your dev environment, and how to scaffold a project using Yeoman. In this third part we will look at how to install and configure some grunt tasks.
- **HTML Tools**: Tools for pre and post processing of the HTML source code.
  - **html-inspector**: HTML Inspector is a code quality tool to help you and your team write better markup. It's written in JavaScript and runs in the browser, so testing your HTML has never been easier.
  - **html-minifier**: HTMLMinifier is a highly configurable, well-tested, Javascript-based HTML minifier, with lint-like capabilities.
- **Image Post Processing**: Tools for image conversion and optimization.
  - **ImageOptim-CLI**: Make lossless optimisation of images part of your automated build process.
  - **Jpegoptim**: Utility to optimize/compress JPEG files.
  - **Optimize Images for Web – Ultimate Guide**: We will discuss the three areas in which you can better optimize images for web: better web performance, rank and index better in search engines, better social media engagement and CTR.
  - **Pngcrush**: Pngcrush is an optimizer for PNG (Portable Network Graphics) files.
  - **SMLR**: Re-encode jpeg images with no perceivable quality loss. Uses the butteraugli psychovisual comparison and k-ary search to determine the best jpeg quality setting.
- **JavaScript Tools**: Tools for static analysis, pre and post processing of JavaScript files.
  - **Babel**: Babel is a generic multi-purpose compiler for JavaScript. Using Babel you can use (and create) the next generation of JavaScript, as well as the next generation of JavaScript tooling.
    - **Full-Stack Redux Tutorial**: We will go through all the steps of constructing a Node+Redux backend and a React+Redux frontend for a real-world application, using test-first development.
    - **JavaScript Transpilers: What They Are & Why We Need Them**: Learn how to use Babel, and what has to do with the future of JavaScript.
  - **Closure Compiler**: The Closure Compiler parses your JavaScript, analyzes it, removes dead code and rewrites and minimizes what's left. It also checks syntax, variable references, and types, and warns about common JavaScript pitfalls.
  - **Flow**: Flow is a static type checker for JavaScript. It can be used to catch common bugs in JavaScript programs before they run.

- **JSCodeshift**: Codemods are tools that assist large-scale, partially automatable codebase refactoring. JSCodeshift is a toolkit for running codemods over multiple JS files.
    - **Turbocharged JavaScript Refactoring with Codemods**: Joe Lencioni describes how they used codemods to transform a large JavaScript code base at AirBnB
- **JavaScript Code Linting**: Linting is the process of running a program that will analyse code for potential errors.
    - **ESLint**: The pluggable linting utility for JavaScript and JSX.
    - **JSHint**: JSHint is a tool for more flexible static analysis of JavaScript programs.
    - **JSLint**: JSLint is a tool for detecting errors or problems by static analysis of JavaScript programs.
    - **JSLint, JSHint and ESLint Error Explanations**: JSLint Error Explanations is designed to help you improve your JavaScript by understanding the sometimes cryptic error messages produced by JSLint, JSHint and ESLint, and teaching you how to avoid such errors.
- **Module Bundlers and Loaders**: Libraries for bundling JavaScript Modules into one or several files.
    - **Browserify**: Browserify lets you require('modules') in the browser by bundling up all of your dependencies.
        - **Budo**: A browserify development server, focused on incremental reloading, LiveReload integration (including CSS injection), and other high-level features.
        - **Watchify**: Watch mode for browserify builds.
    - **CrapLoader**: The goal of crapLoader is to load ads, widgets or any JavaScript code with document.write in it. This library hijacks document.write and delegates the content loaded from each script into the correct position.
    - **Modules Webmake**: A CommonJS module bundler similar to Browserify but much faster due to different requirements finder.
    - **Require.js**: RequireJS is a JavaScript file and AMD module loader. It is optimized for in-browser use, but it can be used in other JavaScript environments.
    - **Require1k**: CommonJS require for the browser in 1KB, with no build needed.
    - **Rollup.js**: Rollup is a next-generation JavaScript module bundler. Author your app or library using ES2015 modules, then efficiently bundle them up into a single file for use in browsers and Node.js.
    - **SystemJS**: Universal dynamic module loader - loads ES6 modules, AMD, CommonJS and global scripts in the browser and NodeJS. Works with both Traceur and Babel.
        - **Modular JavaScript: A Beginners Guide to SystemJS & JSPM**: The combination of jspm and SystemJS provides a unified way of installing and loading dependencies.
    - **URequire**: The Ultimate JavaScript Module Builder & Automagical Task Runner.

        - **Webpack**: Webpack is a module bundler. It takes modules with dependencies and generates static assets representing those modules.
            - **Block, Element, Modifying Your JavaScript Components**: Mark Dalgleish is discussing how to organize React code with BEM and build everything with Webpack.
            - **Developing with Docker and Webpack**: Chris Harrington explains how to create a development environment with Webpack and Docker to match the production as much as possible.
            - **Full-Stack Redux Tutorial**: We will go through all the steps of constructing a Node+Redux backend and a React+Redux frontend for a real-world application, using test-first development.
            - **How to Set Up Webpack Image Loader**: This brief tutorial will help you set up an image loader in Webpack.
            - **The SoundCloud Client in React + Redux**: After finishing this step by step tutorial you will be able to author your own React + Redux project with Webpack and Babel.
            - **Webpack from Apprentice to Master**: The purpose of this guide is to help you get started with Webpack and then go beyond basics.
            - **WebpackBin**: A webpack code sandbox.
            - **Why I think Webpack is the Right Approach To Build Pipelines**: Thomas Boyt compares how Grunt, Gulp, Broccoli and Webpack discover dependencies.
    - **Regenerator**: This package implements a source transformation that takes the proposed syntax for generators/yield from future versions of JS and spits out efficient JS-of-today (ES5) that behaves the same way.
- **Package Management**: A package manager or package management system is a collection of software tools that automates the process of installing, upgrading, configuring, and removing reusable libraries and components in a consistent manner.
    - **Bower**: Bower offers a generic, unopinionated solution to the problem of front-end package management, while exposing the package dependency model via an API that can be consumed by a more opinionated build stack.
    - **Lerna**: Lerna is a tool that optimizes the workflow around managing multi-package repositories with git and npm.
    - **NPM**: NPM makes it easy for JavaScript developers to share and reuse code, and it makes it easy to update the code that you're sharing.
- **Sourcemaps**: Sourcemap is a way to map a combined/minified file back to an unbuilt state.
    - **combine-source-map**: Add source maps of multiple files, offset them and then combine them into one source map.
    - **convert-source-map**: Converts a source-map from/to different formats and allows adding/changing properties.
    - **exorcist**: Externalizes the source map found inside a stream to an external .js.map file

    - **generate-sourcemap**: Generates a source map for files that were packed into a bundle.
    - **inline-source-map**: Adds source mappings and base64 encodes them, so they can be inlined in your generated file.
    - **mold-source-map**: Mold a source map that is almost perfect for you into one that is.
    - **source-map-cjs**: Generates and consumes source maps. Adapted to be commonjs only and work in older browsers.
- **Version Control**: Version control or source control is a system that records changes to a file or set of files over time so that you can recall specific versions later.
    - **Git**: Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.
        - **Become a Git Guru**: A series of Git tutorials by Atlassian.
    - **OctoLinker**: The OctoLinker is a browser extensions which makes references to other files in GitHub clickable.

React Notes:

This appendix is a non-exhaustive list of new syntactic features and methods that were added to JavaScript in ES6. These features are the most commonly used and most helpful.

While this appendix doesn't cover ES6 classes, we go over the basics while learning about components in the book. In addition, this appendix doesn't include descriptions of some larger new features like promises and generators. If you'd like more info on those or on any topic below, we encourage you to reference the Mozilla Developer Network's website MDN.

Prefer `const` and `let` over `var`

If you've worked with ES5 JavaScript before, you're likely used to seeing variables declared with `var` :

```
ar myVariable = 5;
```

Both the `const` and `let` statements also declare variables. They were introduced in ES6.

Use `const` in cases where a variable is never re-assigned. Using `const` makes this clear to whoever is reading your code. It refers to the "constant" state of the variable in the context it is defined within.

If the variable will be re-assigned, use `let` .

We encourage the use of `const` and `let` instead of `var` . In addition to the restriction introduced by `const` , both `const` and `let` are *block scoped* as opposed to *function scoped*. This scoping can help avoid unexpected bugs.

Arrow functions

There are three ways to write arrow function bodies. For the examples below, let's say we have an array of city objects:

```
onst cities = [
  { name: 'Cairo', pop: 7764700 },
  { name: 'Lagos', pop: 8029200 },
];
```

If we write an arrow function that spans multiple lines, we must use braces to delimit the function body like this:

```
const formattedPopulations = cities.map((city) => {
  const popMM = (city.pop / 1000000).toFixed(2);
  return popMM + ' million';
});
console.log(formattedPopulations);
```

Note that we must also explicitly specify a `return` for the function.

However, if we write a function body that is only a single line $or single expression$ we can use parentheses to delimit it:

```
const formattedPopulations2 = cities.map((city) => (
  (city.pop / 1000000).toFixed(2) + ' million'
));
```

Notably, we don't use `return` as it's implied.

Furthermore, if your function body is terse you can write it like so:

```
const pops = cities.map(city => city.pop);
console.log(pops);
```

The terseness of arrow functions is one of two reasons that we use them. Compare the one-liner above to this:

```
const popsNoArrow = cities.map(function(city) { return city.pop });
```

Of greater benefit, though, is how arrow functions bind the `this` object.

The traditional JavaScript function declaration syntax ‘$function()$‘ will bind `this` in anonymous functions to the global object. To illustrate the confusion this causes, consider the following example:

```
unction printSong() {
  console.log("Oops - The Global Object");
}

const jukebox = {
  songs: [
    {
      title: "Wanna Be Startin' Somethin'",
      artist: "Michael Jackson",
    },
    {
      title: "Superstar",
      artist: "Madonna",
    },
  ],
  printSong: function (song) {
    console.log(song.title + " - " + song.artist);
  },
  printSongs: function () {

    this.songs.forEach(function(song) {

      this.printSong(song);
    });
  },
}

jukebox.printSongs();
```

The method `printSongs()` iterates over `this.songs` with `forEach()` . In this context, `this` is bound to the object ‘$jukebox$‘ as expected. However, the anonymous function passed to `forEach()` binds its internal `this` to the global object. As such, `this.printSong(song)` calls the function declared at the top of the example, *not* the method on `jukebox` .

JavaScript developers have traditionally used workarounds for this behavior, but arrow functions solve the problem by **capturing the `this` value of the enclosing context**. Using an arrow function for `printSongs()` has the expected result:

```
  printSongs: function () {
    this.songs.forEach((song) => {

      this.printSong(song);
    });
  },
}

jukebox.printSongs();
```

For this reason, throughout the book we use arrow functions for all anonymous functions.

Modules

ES6 formally supports modules using the `import` / `export` syntax.

Named exports

Inside any file, you can use `export` to specify a variable the module should expose. Here's an example of a file that exports two functions:

```
export const sayHi = () => (console.log('Hi!'));
export const sayBye = () => (console.log('Bye!'));

const saySomething = () => (console.log('Something!'));
```

Now, anywhere we wanted to use these functions we could use `import` . We need to specify which functions we want to import. A common way of doing this is using ES6's destructuring assignment syntax to list them out like this:

```
import { sayHi, sayBye } from './greetings';

sayHi();
sayBye();
```

Importantly, the function that was *not* exported ‘$saySomething$‘ is unavailable outside of the module.

Also note that we supply a **relative path** to `from`, indicating that the ES6 module is a local file as opposed to an npm package.

Instead of inserting an `export` before each variable you'd like to export, you can use this syntax to list off all the exposed variables in one area:

```
const sayHi = () => (console.log('Hi!'));
const sayBye = () => (console.log('Bye!'));

const saySomething = () => (console.log('Something!'));

export { sayHi, sayBye };
```

We can also specify that we'd like to import all of a module's functionality underneath a given namespace with the `import * as <Namespace>` syntax:

```
import * as Greetings from './greetings';

Greetings.sayHi();

Greetings.sayBye();

Greetings.saySomething();
```

Default export

The other type of export is a default export. A module can only contain one default export:

```
const sayHi = () => (console.log('Hi!'));
const sayBye = () => (console.log('Bye!'));

const saySomething = () => (console.log('Something!'));

const Greetings = { sayHi, sayBye };

export default Greetings;
```

This is a common pattern for libraries. It means you can easily import the library wholesale without specifying what individual functions you want:

```
import Greetings from './greetings';

Greetings.sayHi();
Greetings.sayBye();
```

It's not uncommon for a module to use a mix of both named exports and default exports. For instance, with `react-dom` , you can import `ReactDOM` $adefaultexport$ like this:

```
import ReactDOM from 'react-dom';

ReactDOM.render(

);
```

Or, if you're only going to use the `render()` function, you can import the named `render()` function like this:

```
import { render } from 'react-dom';

render(

);
```

To achieve this flexibility, the export implementation for `react-dom` looks something like this:

```
export const render = (component, target) => {

};

const ReactDOM = {
  render,

};

export default ReactDOM;
```

If you want to play around with the module syntax, check out the folder `code/webpack/es6-modules` .

For more reading on ES6 modules, see this article from Mozilla: "ES6 in Depth: Modules".

`Object.assign()`

We use `Object.assign()` often throughout the book. We use it in areas where we want to create a modified version of an existing object.

`Object.assign()` accepts any number of objects as arguments. When the function receives two arguments, it *copies* the properties of the second object onto the first, like so:

```
onst coffee = { };
const noCream = { cream: false };
const noMilk = { milk: false };
Object.assign(coffee, noCream);
```

It is idiomatic to pass in three arguments to `Object.assign()`. The first argument is a new JavaScript object, the one that `Object.assign()` will ultimately return. The second is the object whose properties we'd like to build off of. The last is the changes we'd like to apply:

```
const coffeeWithMilk = Object.assign({}, coffee, { milk: true });
```

`Object.assign()` is a handy method for working with "immutable" JavaScript objects.

Template literals

In ES5 JavaScript, you'd interpolate variables into strings like this:

```
var greeting = 'Hello, ' + user + '! It is ' + degF + ' degrees outside.';
```

With ES6 template literals, we can create the same string like this:

```
const greeting = `Hello, ${user}! It is ${degF} degrees outside.`;
```

The spread operator '...'

In arrays, the ellipsis `...` operator will *expand* the array that follows into the parent array. The spread operator enables us to succinctly construct new arrays as a composite of existing arrays.

Here is an example:

```
onst a = [ 1, 2, 3 ];
const b = [ 4, 5, 6 ];
const c = [ ...a, ...b, 7, 8, 9 ];

console.log(c);
```

Notice how this is different than if we wrote:

```
const d = [ a, b, 7, 8, 9 ];
console.log(d);
```

Enhanced object literals

In ES5, all objects were required to have explicit key and value declarations:

```
const explicit = {
  getState: getState,
  dispatch: dispatch,
};
```

In ES6, you can use this terser syntax whenever the property name and variable name are the same:

```
const implicit = {
  getState,
  dispatch,
};
```

Lots of open source libraries use this syntax, so it's good to be familiar with it. But whether you use it in your own code is a matter of stylistic preference.

Default arguments

With ES6, you can specify a default value for an argument in the case that it is `undefined` when the function is called.

This:

```
unction divide(a, b) {

  const divisor = typeof b === 'undefined' ? 1 : b;

  return a / divisor;
}
```

Can be written as this:

```
function divide(a, b = 1) {
  return a / b;
}
```

In both cases, using the function looks like this:

```
divide(14, 2);

divide(14, undefined);

divide(14);
```

Whenever the argument `b` in the example above is `undefined` , the default argument is used. Note that `null` will not use the default argument:

```
divide(14, null);
```

Destructuring assignments

For arrays

In ES5, extracting and assigning multiple elements from an array looked like this:

```
ar fruits = [ 'apples', 'bananas', 'oranges' ];
var fruit1 = fruits[0];
var fruit2 = fruits[1];
```

In ES6, we can use the destructuring syntax to accomplish the same task like this:

```
const [ veg1, veg2 ] = [ 'asparagus', 'broccoli', 'onion' ];
console.log(veg1);
console.log(veg2);
```

The variables in the array on the left are "matched" and assigned to the corresponding elements in the array on the right. Note that `'onion'` is ignored and has no variable bound to it.

For objects

We can do something similar for extracting object properties into variables:

```
const smoothie = {
  fats: [ 'avocado', 'peanut butter', 'greek yogurt' ],
  liquids: [ 'almond milk' ],
  greens: [ 'spinach' ],
  fruits: [ 'blueberry', 'banana' ],
};

const { liquids, fruits } = smoothie;

console.log(liquids);
console.log(fruits);
```

Parameter context matching

We can use these same principles to bind arguments inside a function to properties of an object supplied as an argument:

```
const containsSpinach = ({ greens }) => {
  if (greens.find(g => g === 'spinach')) {
    return true;
  } else {
    return false;
  }
};

containsSpinach(smoothie);
```

We do this often with functional React components:

```
const IngredientList = ({ ingredients, onClick }) => (
  <ul className='IngredientList'>
    {
      ingredients.map(i => (
        <li
          key={i.id}
          onClick={() => onClick(i.id)}
          className='item'
        >
          {i.name}
        </li>
      ))
    }
  </ul>
)
```

Here, we use destructuring to extract the props into variables ‘*ingredients*‘*and*‘*onClick*‘ that we then use inside the component's function body.

Components And Props

Components and Props

Components let you split the UI into independent, reusable pieces, and think about each piece in isolation. This page provides an introduction to the idea of components. You can find a detailed component API reference here.

Conceptually, components are like JavaScript functions. They accept arbitrary inputs *called*“*props*” and return React elements describing what should appear on the screen.

Function and Class Components

The simplest way to define a component is to write a JavaScript function:

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}
```

This function is a valid React component because it accepts a single “props” *whichstandsforproperties* object argument with data and returns a React element. We call such components “function components” because they are literally JavaScript functions.

You can also use an ES6 class to define a component:

```
class Welcome extends React.Component {
  render() {
    return <h1>Hello, {this.props.name}</h1>;
  }
}
```

The above two components are equivalent from React’s point of view.

Function and Class components both have some additional features that we will discuss in the next sections.

Rendering a Component

Previously, we only encountered React elements that represent DOM tags:

```
const element = <div />;
```

However, elements can also represent user-defined components:

```
const element = <Welcome name="Sara" />;
```

When React sees an element representing a user-defined component, it passes JSX attributes and children to this component as a single object. We call this object “props”.

For example, this code renders “Hello, Sara” on the page:

```
function Welcome(props) {  return <h1>Hello, {props.name}</h1>;
}

const element = <Welcome name="Sara" />;ReactDOM.render(
  element,
  document.getElementById('root')
);
```

Try it on CodePen

Let’s recap what happens in this example:

1. We call `ReactDOM.render()` with the `<Welcome name="Sara" />` element.
2. React calls the `Welcome` component with `{name: 'Sara'}` as the props.
3. Our `Welcome` component returns a `<h1>Hello, Sara</h1>` element as the result.

4. React DOM efficiently updates the DOM to match `<h1>Hello, Sara</h1>`.

> **Note:** Always start component names with a capital letter.
>
> React treats components starting with lowercase letters as DOM tags. For example, `<div />` represents an HTML div tag, but `<Welcome />` represents a component and requires `Welcome` to be in scope.
>
> To learn more about the reasoning behind this convention, please read JSX In Depth.

Composing Components

Components can refer to other components in their output. This lets us use the same component abstraction for any level of detail. A button, a form, a dialog, a screen: in React apps, all those are commonly expressed as components.

For example, we can create an `App` component that renders `Welcome` many times:

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}

function App() {
  return (
    <div>
      <Welcome name="Sara" />      <Welcome name="Cahal" />      <Welcome name="Edite" />    </c
  );
}

ReactDOM.render(
  <App />,
  document.getElementById('root')
);
```

Try it on CodePen

Typically, new React apps have a single `App` component at the very top. However, if you integrate React into an existing app, you might start bottom-up with a small component like `Button` and gradually work your way to the top of the view hierarchy.

Extracting Components

Don’t be afraid to split components into smaller components.

For example, consider this `Comment` component:

```
function Comment(props) {
  return (
    <div className="Comment">
      <div className="UserInfo">
        <img className="Avatar"
          src={props.author.avatarUrl}
          alt={props.author.name}
        />
        <div className="UserInfo-name">
          {props.author.name}
        </div>
      </div>
      <div className="Comment-text">
        {props.text}
      </div>
      <div className="Comment-date">
        {formatDate(props.date)}
      </div>
    </div>
  );
}
```

Try it on CodePen

It accepts `author` $anobject$, `text` $astring$, and `date` $adate$ as props, and describes a comment on a social media website.

This component can be tricky to change because of all the nesting, and it is also hard to reuse individual parts of it. Let’s extract a few components from it.

First, we will extract `Avatar` :

```
function Avatar(props) {
  return (
    <img className="Avatar"      src={props.user.avatarUrl}      alt={props.user.name}    />  );
}
```

The `Avatar` doesn’t need to know that it is being rendered inside a `Comment` . This is why we have given its prop a more generic name: `user` rather than `author` .

We recommend naming props from the component’s own point of view rather than the context in which it is being used.

We can now simplify `Comment` a tiny bit:

```
function Comment(props) {
  return (
    <div className="Comment">
      <div className="UserInfo">
        <Avatar user={props.author} />        <div className="UserInfo-name">
          {props.author.name}
        </div>
      </div>
      <div className="Comment-text">
        {props.text}
      </div>
      <div className="Comment-date">
        {formatDate(props.date)}
      </div>
    </div>
  );
}
```

Next, we will extract a `UserInfo` component that renders an `Avatar` next to the user's name:

```
function UserInfo(props) {
  return (
    <div className="UserInfo">      <Avatar user={props.user} />      <div className="UserInfo-r
}
```

This lets us simplify `Comment` even further:

```
function Comment(props) {
  return (
    <div className="Comment">
      <UserInfo user={props.author} />      <div className="Comment-text">
        {props.text}
      </div>
      <div className="Comment-date">
        {formatDate(props.date)}
      </div>
    </div>
  );
}
```

Try it on CodePen

Extracting components might seem like grunt work at first, but having a palette of reusable components pays off in larger apps. A good rule of thumb is that if a part of your UI is used several times ‘$Button$‘, ‘$Panel$‘, ‘$Avatar$‘, or is complex enough on its own ‘App‘, ‘$FeedStory$‘, ‘$Comment$‘, it is a good candidate to be extracted to a separate component.

Props are Read-Only

Whether you declare a component as a function or a class, it must never modify its own props. Consider this `sum` function:

```
function sum(a, b) {
  return a + b;
}
```

Such functions are called "pure" because they do not attempt to change their inputs, and always return the same result for the same inputs.

In contrast, this function is impure because it changes its own input:

```
function withdraw(account, amount) {
  account.total -= amount;
}
```

React is pretty flexible but it has a single strict rule:

All React components must act like pure functions with respect to their props.

Of course, application UIs are dynamic and change over time. In the next section, we will introduce a new concept of "state". State allows React components to change their output over time in response to user actions, network responses, and anything else, without violating this rule.

Composition vs Inheritance

React has a powerful composition model, and we recommend using composition instead of inheritance to reuse code between components.

In this section, we will consider a few problems where developers new to React often reach for inheritance, and show how we can solve them with composition.

Containment

Some components don't know their children ahead of time. This is especially common for components like `Sidebar` or `Dialog` that represent generic "boxes".

We recommend that such components use the special `children` prop to pass children elements directly into their output:

```
function FancyBorder(props) {
  return (
    <div className={'FancyBorder FancyBorder-' + props.color}>
      {props.children}    </div>
  );
}
```

This lets other components pass arbitrary children to them by nesting the JSX:

```
function WelcomeDialog() {
  return (
    <FancyBorder color="blue">
      <h1 className="Dialog-title">        Welcome      </h1>      <p className="Dialog-message"
  );
}
```

Try it on CodePen

Anything inside the `<FancyBorder>` JSX tag gets passed into the `FancyBorder` component as a `children` prop. Since `FancyBorder` renders `{props.children}` inside a `<div>` , the passed elements appear in the final output.

While this is less common, sometimes you might need multiple “holes” in a component. In such cases you may come up with your own convention instead of using `children` :

```
function SplitPane(props) {
  return (
    <div className="SplitPane">
      <div className="SplitPane-left">
        {props.left}      </div>
      <div className="SplitPane-right">
        {props.right}      </div>
    </div>
  );
}

function App() {
  return (
    <SplitPane
      left={
        <Contacts />      }
      right={
        <Chat />      } />
  );
}
```

Try it on CodePen

React elements like `<Contacts />` and `<Chat />` are just objects, so you can pass them as props like any other data. This approach may remind you of “slots” in other libraries but there are no limitations on what you can pass as props in React.

Specialization

Sometimes we think about components as being “special cases” of other components. For example, we might say that a `WelcomeDialog` is a special case of `Dialog` .

In React, this is also achieved by composition, where a more “specific” component renders a more “generic” one and configures it with props:

```
function Dialog(props) {
  return (
    <FancyBorder color="blue">
      <h1 className="Dialog-title">
        {props.title}      </h1>
      <p className="Dialog-message">
        {props.message}      </p>
    </FancyBorder>
  );
}

function WelcomeDialog() {
  return (
    <Dialog      title="Welcome"      message="Thank you for visiting our spacecraft!" />  );
}
```

Try it on CodePen

Composition works equally well for components defined as classes:

```
function Dialog(props) {
  return (
    <FancyBorder color="blue">
      <h1 className="Dialog-title">
        {props.title}
      </h1>
      <p className="Dialog-message">
        {props.message}
      </p>
      {props.children}    </FancyBorder>
  );
}

class SignUpDialog extends React.Component {
  constructor(props) {
    super(props);
    this.handleChange = this.handleChange.bind(this);
    this.handleSignUp = this.handleSignUp.bind(this);
    this.state = {login: ''};
  }

  render() {
    return (
      <Dialog title="Mars Exploration Program"
              message="How should we refer to you?">
        <input value={this.state.login}               onChange={this.handleChange} />        <bu
    );
  }

  handleChange(e) {
    this.setState({login: e.target.value});
  }

  handleSignUp() {
    alert(`Welcome aboard, ${this.state.login}!`);
  }
}
```

Try it on CodePen

So What About Inheritance?

At Facebook, we use React in thousands of components, and we haven’t found any use cases where we would recommend creating component inheritance hierarchies.

Props and composition give you all the flexibility you need to customize a component’s look and behavior in an explicit and safe way. Remember that components may accept arbitrary props, including primitive values, React elements, or functions.

If you want to reuse non-UI functionality between components, we suggest extracting it into a separate JavaScript module. The components may import it and use that function, object, or a class, without extending it.

downloads

{% file src="../.gitbook/assets/components-and-props-react.md" caption="componentsandProps" %}

{% file src="../.gitbook/assets/lifting-state-up-react.md" caption="State" %}

{% file src="../.gitbook/assets/jsx-in-depth-react 1.md" caption="JSX" %}

Hello World

Hello World

The smallest React example looks like this:

```
ReactDOM.render(
  <h1>Hello, world!</h1>,
  document.getElementById('root')
);
```

It displays a heading saying “Hello, world!” on the page.

Try it on CodePen

Click the link above to open an online editor. Feel free to make some changes, and see how they affect the output. Most pages in this guide will have editable examples like this one.

How to Read This Guide

In this guide, we will examine the building blocks of React apps: elements and components. Once you master them, you can create complex apps from small reusable pieces.

> Tip
>
> This guide is designed for people who prefer **learning concepts step by step**. If you prefer to learn by doing, check out our practical tutorial. You might find this guide and the tutorial

> complementary to each other.

This is the first chapter in a step-by-step guide about main React concepts. You can find a list of all its chapters in the navigation sidebar. If you're reading this from a mobile device, you can access the navigation by pressing the button in the bottom right corner of your screen.

Every chapter in this guide builds on the knowledge introduced in earlier chapters. **You can learn most of React by reading the "Main Concepts" guide chapters in the order they appear in the sidebar.** For example, "Introducing JSX" is the next chapter after this one.

Knowledge Level Assumptions

React is a JavaScript library, and so we'll assume you have a basic understanding of the JavaScript language. **If you don't feel very confident, we recommend going through a JavaScript tutorial to check your knowledge level** and enable you to follow along this guide without getting lost. It might take you between 30 minutes and an hour, but as a result you won't have to feel like you're learning both React and JavaScript at the same time.

> Note
>
> This guide occasionally uses some of the newer JavaScript syntax in the examples. If you haven't worked with JavaScript in the last few years, these three points should get you most of the way.

JSX

Introducing JSX

Consider this variable declaration:

```
const element = <h1>Hello, world!</h1>;
```

This funny tag syntax is neither a string nor HTML.

It is called JSX, and it is a syntax extension to JavaScript. We recommend using it with React to describe what the UI should look like. JSX may remind you of a template language, but it comes with the full power of JavaScript.

JSX produces React "elements". We will explore rendering them to the DOM in the next section. Below, you can find the basics of JSX necessary to get you started.

Why JSX?

React embraces the fact that rendering logic is inherently coupled with other UI logic: how events are handled, how the state changes over time, and how the data is prepared for display.

Instead of artificially separating *technologies* by putting markup and logic in separate files, React separates *concerns* with loosely coupled units called “components” that contain both. We will come back to components in a further section, but if you’re not yet comfortable putting markup in JS, this talk might convince you otherwise.

React doesn’t require using JSX, but most people find it helpful as a visual aid when working with UI inside the JavaScript code. It also allows React to show more useful error and warning messages.

With that out of the way, let’s get started!

Embedding Expressions in JSX

In the example below, we declare a variable called `name` and then use it inside JSX by wrapping it in curly braces:

```
const name = 'Josh Perez';const element = <h1>Hello, {name}</h1>;
ReactDOM.render(
  element,
  document.getElementById('root')
);
```

You can put any valid JavaScript expression inside the curly braces in JSX. For example, `2 + 2`, `user.firstName`, or `formatName(user)` are all valid JavaScript expressions.

In the example below, we embed the result of calling a JavaScript function, `formatName(user)`, into an `<h1>` element.

```
function formatName(user) {
  return user.firstName + ' ' + user.lastName;
}

const user = {
  firstName: 'Harper',
  lastName: 'Perez'
};

const element = (
  <h1>
    Hello, {formatName(user)}!  </h1>
);

ReactDOM.render(
  element,
  document.getElementById('root')
);
```

Try it on CodePen

We split JSX over multiple lines for readability. While it isn't required, when doing this, we also recommend wrapping it in parentheses to avoid the pitfalls of automatic semicolon insertion.

JSX is an Expression Too

After compilation, JSX expressions become regular JavaScript function calls and evaluate to JavaScript objects.

This means that you can use JSX inside of `if` statements and `for` loops, assign it to variables, accept it as arguments, and return it from functions:

```
function getGreeting(user) {
  if (user) {
    return <h1>Hello, {formatName(user)}!</h1>;  }
  return <h1>Hello, Stranger.</h1>;}
```

Specifying Attributes with JSX

You may use quotes to specify string literals as attributes:

```
const element = <div tabIndex="0"></div>;
```

You may also use curly braces to embed a JavaScript expression in an attribute:

```
const element = <img src={user.avatarUrl}></img>;
```

Don’t put quotes around curly braces when embedding a JavaScript expression in an attribute. You should either use quotes $forstringvalues$ or curly braces $forexpressions$, but not both in the same attribute.

> **Warning:**
>
> Since JSX is closer to JavaScript than to HTML, React DOM uses `camelCase` property naming convention instead of HTML attribute names.
>
> For example, `class` becomes `className` in JSX, and `tabindex` becomes `tabIndex` .

Specifying Children with JSX

If a tag is empty, you may close it immediately with `/>` , like XML:

```
const element = <img src={user.avatarUrl} />;
```

JSX tags may contain children:

```
const element = (
  <div>
    <h1>Hello!</h1>
    <h2>Good to see you here.</h2>
  </div>
);
```

JSX Prevents Injection Attacks

It is safe to embed user input in JSX:

```
const title = response.potentiallyMaliciousInput;
// This is safe:
const element = <h1>{title}</h1>;
```

By default, React DOM escapes any values embedded in JSX before rendering them. Thus it ensures that you can never inject anything that’s not explicitly written in your application. Everything is converted to a string before being rendered. This helps prevent XSS $cross-site-scripting$ attacks.

JSX Represents Objects

Babel compiles JSX down to `React.createElement()` calls.

These two examples are identical:

```
const element = (
  <h1 className="greeting">
    Hello, world!
  </h1>
);
```

```
const element = React.createElement(
  'h1',
  {className: 'greeting'},
  'Hello, world!'
);
```

`React.createElement()` performs a few checks to help you write bug-free code but essentially it creates an object like this:

```
// Note: this structure is simplified
const element = {
  type: 'h1',
  props: {
    className: 'greeting',
    children: 'Hello, world!'
  }
};
```

These objects are called “React elements”. You can think of them as descriptions of what you want to see on the screen. React reads these objects and uses them to construct the DOM and keep it up to date.

We will explore rendering React elements to the DOM in the next section.

> **Tip:**
>
> We recommend using the “Babel” language definition for your editor of choice so that both ES6 and JSX code is properly highlighted.

REACT ENVIORMENT

First thing we need to do is setup our development environment and hook up React. Our end goal will be to get a basic message up on screen. Let's create a new directory and download the packages we need.

```
mkdir starwars_api
cd starwars_api
npm init -y
npm install --save react react-dom redux react-redux redux-thunk
npm install --save-dev babel-core babel-preset-es2015 babel-preset-react webpack@2.1.0-beta.27 f
```

Now let's create our entrance file. Create a file at `src/app.js` and put the following in it.

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './components/App';
import { AppContainer } from 'react-hot-loader';

require('./index.html');

// Create app
const container = document.querySelector('#app-container');

// Render app
ReactDOM.render(
  <AppContainer>
    <App />
  </AppContainer>
  , container
);

// Hot module reloading
if (module.hot) {
  module.hot.accept('./components/App', () => {
    ReactDOM.render(
      <AppContainer>
        <App />
      </AppContainer>
      , container
    );
  });
}
```

We are displaying a central `App` component. So let's create it! Create a file at `src/components/App.js` and put the following in it.

```
import React from 'react';

const App = () =>
  <div className='container'>
    <div className='row'>
      Hello, World!
    </div>
  </div>;

export default App;
```

We also need to create our `index.html` file that we are including in our `app.js` file. Create a file at `src/index.html` and put the following in it.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>React Star Wars</title>

  <!-- Latest compiled and minified CSS -->
  <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.mir

  <!-- Optional theme -->
  <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap-the

  <style>
    h1 {
      font-size: 30px;
      font-weight: 200;
    }

    h2 {
      font-size: 22px;
      margin-top: 5px;
    }

    ul {
      padding-left: 15px;
    }
  </style>
</head>
<body>
  <div id="app-container"></div>
  <script src="app.bundle.js"></script>
</body>
</html>
```

This is a basic HTML page. We are including Bootstrap and its basic theme for some simple styles. We also are adding some styles in a `style` tag to help with the sizing of some elements. In the `body` , we are adding our container element and our main app bundle.

Now that we have all that setup, let's create a webpack config that we can use to build our project. Create a file in the root of your project called `webpack.config.js` and put the following in it.

```
const webpack = require('webpack');
const path = require('path');

module.exports = {
  entry: [
    'react-hot-loader/patch',
    './src/app.js',
  ],
  output: {
    path: path.resolve(__dirname, './build'),
    filename: 'app.bundle.js',
  },
  module: {
    loaders: [
      {
        test: /\.html$/,
        loader: 'file-loader?name=[name].[ext]',
      },
      {
        test: /\.jsx?$/,
        exclude: /node_modules/,
        loader: 'babel-loader',
      },
    ],
  },
  plugins: [
    new webpack.NamedModulesPlugin(),
  ],
};
```

Here we are exporting our config object like normal. We point webpack to start at our `app.js` file and tell everything to output to a `build` directory. We run all HTML files though the file loader. This essentially just moves the file to our `build` directory. Also, we run our JS files through the babel loader to make sure they are transpiled to code that most browsers these days understand. Lastly, we are including a plugin that will give us pretty output in the console.

Let's create a `.babelrc` file to configure babel.

```
{
  "presets": [
    ["es2015", { "modules": false }],
    "react"
  ],
  "plugins": [
    "react-hot-loader/babel"
  ]
}
```

Last thing. Let's create some npm scripts so that we don't need to remember the commands we need to run our application. Add the following scripts to your `package.json` file in the `scripts` section.

React-Resources

Awesome React

A collection of awesome things regarding the React ecosystem.

- React
  - React General Resources
  - React Community
  - React Online Playgrounds
  - React Tutorials
    - React General Tutorials
    - React Hooks
    - React and TypeScript
    - React Performance
    - React Internals
    - React Interview Questions
  - React Tools
    - React Development Tools
    - React Frameworks
    - React Styling
    - React Routing
    - React Component Libraries
    - React Awesome Components
    - React Testing
    - React Libraries

    - ReactRally Conf 2016 Playlist
    - React.js Amsterdam 2018 Playlist
    - Video Tutorials
- Demo React Apps
- Real React Apps
- Contribution

React

> JavaScript Library for building User Interfaces

React General Resources

- React Official Website
- React Documentation
- React GitHub

React Community

- Reactiflux Discord Channel
- React StackOverflow
- React Twitter

React Online Playgrounds

- CodePen
- CodeSandbox
- JSFiddle

Another Awesome Lists

- React/Redux Links

React Tutorials

React General Tutorials

- React Official Tutorial
- Using React in Visual Studio Code
- Scrimba - Learn React for free interactively
- FreeCodeCamp React Challenges
- React Cheatsheet
- React Patterns

- Setup Flow with React

React Hooks

- React Hooks
- Awesome React Hooks
- Thinking in React Hooks
- Replacing Redux with React Hooks and Context
- React Hooks cheat sheet: Unlock solutions to common problems
- How to fetch data with React Hooks?
- Easy to understand React Hook recipes
- React Hooks Video Tutorial

React and TypeScript

- TypeScript, React and Webpack
- JSX in TypeScript
- Cheatsheets for experienced React developers getting started with TypeScript
- React by Example

React Performance

- React Optimizing Performance
- Introducing the React Profiler
- Optimizing React: Virtual DOM explained
- A Definitive Guide to Optimize Major Performance issues in React
- Twitter Lite and High Performance React Progressive Web Apps at Scale
- Using the React DevTools Profiler to Diagnose React App Performance Issues
- Top 5 Practices to Boost React Performance
- React is Slow, React is Fast: Optimizing react Apps in Practice
- Rendering large lists with react-window

React Internals

- Reconciliation
- React Fiber Architecture
- Build your own React
- Inside Fiber: In-depth overview of the new reconciliation algorithm in React
- Entire React code base explanation by visual block schemes

React Interview Questions

- 13 Essential React Interview Questions
- List of React interview Questions and Answers
- React Coding Challenges

React Tools

React Development Tools

- react-devtools - Inspection of React component hierarchy in the Chrome and Firefox Developer Tools
- react-hot-loader - Tweak React components in real time
- react-loadable - A higher order component for loading components with promises
- loadable-components - React code splitting made easy
- reactotron - A desktop app for inspecting your React and React Native projects
- storybook - UI component dev & test
- docz - Zero Config, live-reloading documentation with Markdown + JSX
- react-styleguidist - Isolated React component development environment with a living style guide
- react-cosmos - Dev tool for creating reusable React components
- eslint-plugin-react - React specific linting rules for ESLint
- eslint-plugin-jsx-a11y - Static AST checker for a11y rules on JSX elements
- @axe-core/react - Accessibility auditing for React applications
- DataFormsJS JSX Loader - Small JavaScript Compiler for quickly converting JSX to JS directly on a web page
- Why Did You Render - Monkey patches React to notify you about avoidable re-renders.
- Divjoy - React codebase and UI generator to speed up development $paid$
- Plasmic - Powerful design tool for building your React components visually.

React Starter Kits and Toolchains

- create-react-app - Set up a modern Web app by running one command
- Razzle - Build production ready React applications. Razzle is toolchain for modern static and dynamic websites and web applications
- Neutrino React Preset - `@neutrinojs/react` is a Neutrino preset that supports building React web applications
- react-starter-kit - Isomorphic Web app boilerplate
- create-react-library - CLI for creating reusable, modern React libraries using Rollup and create-react-app.
- tsdx - Zero-config CLI for TypeScript package development

React Frameworks

- next.js - The React Framework
- gatsby.js - Free and open source framework based on React
- react-admin - Frontend Framework for building B2B applications on top of REST/GraphQL APIs
- remix - Finally, a killer React framework from the creators of React Router
- Blitz - The Fullstack React Framework
- aleph.js - The React Framework in Deno

React Styling

- styled-components - Visual primitives for the component age
- emotion - Library designed for writing CSS styles with JavaScript
- radium - A toolchain for React component styling
- jss - Authoring tool for CSS
- aphrodite - Framework-agnostic CSS-in-JS with support for server-side rendering, browser prefixing, and minimum CSS generation

React Routing

- react-router - Declarative routing for React
- navi - Declarative, asynchronous routing for React
- curi - JavaScript router for single-page applications
- reach - Next Generation Routing for React
- universal-router - A simple middleware-style router for isomorphic JavaScript web apps
- wouter - A minimalist-friendly ~1.3KB routing library

React Component Libraries

- material-ui - React components for faster and easier web development
- blueprint - A React-based UI toolkit for the webs
- Fluent UI - A set of React components for building Microsoft web experiences
- react-bootstrap - Bootstrap components built with React
- reactstrap - Simple React Bootstrap 4 components
- ant-design - A design system with values of Nature and Determinacy
- chakra-ui - Simple, Modular & Accessible UI Components for your React Applications
- semantic-ui-react - The official Semantic-UI-React integration
- evergreen - Evergreen React UI Framework by Segment
- grommet - A react-based framework that provides accessibility, modularity, responsiveness, and theming in a tidy package
- rebass - React primitive UI components built with styled-system
- reakit - Accessible, Composable and Customizable components for React
- rsuite - A suite of React components

- atlaskit - Atlassian's official UI library, built according to the Atlassian Design Guidelines.
- baseweb - Base Web is a foundation for initiating, evolving, and unifying web products.
- primereact - A complete UI Framework for React with 50+ components featuring material, bootstrap and custom themes.
- eui - Elastic UI Framework
- react-spectrum - Adobe's collection of libraries and tools that help you build adaptive, accessible, and robust user experiences
- ring-ui - JetBrains Web UI components
- react-bulma-components - React components for Bulma framework
- react-bulma - React.js components for Modern CSS framework based on Flexbox
- trunx - Super Saiyan React components, son of awesome Bulma, implemented in TypeScript
- bumbag-ui - Build accessible & themeable React applications with your Bumbag

React Awesome Components

- Awesome React Components list
- react-select - The Select Component for React
- react-beautiful-dnd - Beautiful and accessible drag and drop for lists with React
- react-dnd - Drag and Drop for React
- react-grid-layout - A draggable and resizable grid layout with responsive breakpoints
- react-table - A lightweight, fast and extendable datagrid for React
- react-data-grid - Excel-like grid component built with React
- react-draggable - React draggable component
- react-resizable-and-movable - A resizable and draggable component for React
- react-resizable - A simple React component that is resizable with a handle
- react-resizable-box - A resizable component for React
- react-searchbox-awesome - Minimalistic searchbox
- react-sortable-pane - A sortable and resizable pane component for React
- react-spaces - Nestable resizable, anchored, scrollable components
- react-dates - An easily internationalizable, mobile-friendly datepicker library for the web
- react-big-calendar - Calendar component
- react-datepicker - ReactJS Datepicker
- react-list - A versatile infinite scroll React component
- react-intl - Internationalize React apps
- react-i18next - Internationalization for React done right
- react-aria-modal - A fully accessible React modal
- react-hotkeys - Declarative hotkey and focus area management for React
- react-keydown - Lightweight keydown wrapper for React components
- react-joyride - Create guided tours for your apps

- react-virtualized - React components for efficiently rendering large lists and tabular data
- react-window - React components for efficiently rendering large lists and tabular data
- react-text-mask - Input mask for React
- react-loading-skeleton - Create skeleton screens that automatically adapt to your app
- react-spinkit - A collection of loading indicators animated with CSS for React
- rheostat - Accessible slider component built with React
- qrcode.react - QR component for use with React
- react-archer - Draw arrows between React elements
- react-pdf-viewer - A PDF viewer made for React
- react-parallax-tilt - Easily apply tilt hover effect on React components
- react-popper - Position tooltips and popovers in an elegant, performant manner
- react-tsparticles - Easily create highly customizable particles animations
- react-spring - Spring-physics based animation library for React applications
- framer-motion - A React library to power production-ready animations
- react-accessible-accordion - React Component for creating an 'Accordion' that adheres to the WAI ARIA spec for accessibility.
- react-truncate-markup - React component for truncating JSX markup.
- react-cookie - Universal cookies for React
- react-slick - Carousel component built with React
- react-gtm-module - Google Tag Manager Module for React
- react-device-detect - Detect device for React
- react-colorful - A tiny $2,5KB$, dependency-free, fast and accessible color picker component
- react-modal - Accessible modal dialog component for React
- cleave.js - Format input text content when you are typing
- react-fontawesome - Font Awesome 5 React component

React Testing

- jest - Delightful JavaScript Testing Framework
- enzyme - JavaScript Testing utilities for React
- react-testing-library - Simple and complete React DOM testing utilities
- react-hooks-testing-library - React hooks testing utilities that encourage good testing practices
- majestic - Zero config GUI for Jest

React Libraries

- react-border-wrapper - A wrapper for placing elements along div borders in React.
- react-magic - Automatically AJAXify plain HTML with the power of React
- react-toolbox - A set of React components implementing Google's Material Design specification
- tcomb-react - Library allowing you to check all the props of your React components

- react-responsive - Media queries in react for responsive design
- preact - Fast 3kb React alternative with the same ES6 API.
- riotjs - A React-like, 3.5KB user interface library
- Maple.js - Bringing the concept of web-components to React
- react-i13n - A performant, scalable and pluggable approach to instrumenting your React application
- react-icons - svg react icons of popular icon packs
- react-open-doodles - Awesome free illustrations as react components.
- Keo - Plain functions for a more functional Deku approach to creating React components, with functional goodies such as pipe, memoize, etc...
- Bit - A virtual repository for managing and using react and other web components across applications
- AtlasKit - Atlassian's React UI library
- ReactiveSearch - UI components library for Elasticsearch
- Slate - A completely customizable framework for building rich text editors.
- react-json-schema - Construct React elements from JSON by mapping JSON definitions to React components that you expose.
- react-lodash - Lodash as React components
- react-helmet - A document head manager for React
- react-snap - Zero-configuration framework-agnostic static prerendering for SPAs
- Draft.js - A React framework for building text editors
- refract - Harness the power of reactive programming to supercharge your components
- react-desktop - OS X and Windows UI components built with React
- reapop - A simple and customizable React notifications system
- react-extras - Useful components and utilities for working with React
- react-instantsearch - Lightning-fast search for React and React Native applications, by Algolia
- uppy - The next open source file uploader for web browsers
- react-motion - A spring that solves your animation problems
- react-esi - React Edge Side Includes
- react-aria - Adobe's library of React Hooks that provides accessible UI primitives for your design system
- react-uploady - Modern file-upload components & hooks for React.

React Integration

- ReasonReact
- React Rails
- ReactJS. NET
- om - ClojureScript interface

- Reagent - A minimalistic ClojureScript interface to React.js
- Express React views
- React Page Middleware
- ngReact - React Components in Angular
- coffee-react-transform - Provides React JSX support for Coffeescript
- sprockets-coffee-react - Sprockets preprocessor for CJSX
- react-kup - A simple, non-intrusive alternative to jsx for coffeescript
- turbo-react - Combine Turbolinks and React to apply DOM diffs
- react-bacon - A little module for using React with Bacon.js
- msx - React's JSX Transformer, tweaked to output calls to Mithril
- react-backbone - Backbone-aware mixins for react
- NestedReact - transparent integration with Backbone Views and NestedTypes models
- backbone-reaction - React, Backbone and then some
- react.backbone - Plugin for React to make Backbone migration easier
- reactbone - React extensions for Backbone
- backbone-react-ui - React components for use with backbone and backbone paginator
- react-events - Declarative managed event bindings for react components
- react-mixin-manager - React mixin registration manager
- react-topcoat by @plaxdan - Topcoat CSS components built with the React library
- react-topcoat by @arnemart - A collection of React components for Topcoat
- reactdown - Write React components using markdown syntax
- react-jade - Compile Jade to React JavaScript
- jade-react - Compile Jade templates to React. DOM expressions
- gulp-jade-react - Compile Jade templates into React de-sugared JSX with Gulp
- sbt-reactjs - React SBT Plugin using npm
- scalajs-react - A guilty affair between Scala.js and Facebook's React
- react-xtags - Using React to implement xtags
- jreact - React on server-side Java $withRhinoorNashorn$
- React.hiccup - A complete replacement for JSX written in sweet.js
- react-play - Rendering React components in the Play Framework with JDK8's Nashorn
- rx-react - Utilities to works with React in a RxJS
- react-with-di - A hacked prototype of React.js with DI
- reactfire - ReactJS mixin for easy Firebase integration
- react-clickdrag-mixin - ClickDrag mixin for React component
- react-masonry-mixin - Standalone mixin for Masonry $@desandro$
- react-packery-mixin - Standalone mixin for Packery $Metafizzy$
- react-dropzone - Simple HTML5 drag-drop zone with React.js.
- aframe-react - A-Frame VR + React
- react-three-fiber - A react reconciler for threejs $webandreact-native$

- react-three - React bindings to create and control a 3D scene using three.js
- react-three-renderer - Render into a three.js canvas using React
- react-threejs - Simplest bindings between React & Three.js
- react-masonry-css - Fast Masonry layout powered by CSS, dependency free
- react-captcha - A react.js reCAPTCHA for Google
- reaptcha - Clean, modern and simple React wrapper for Google reCAPTCHA
- react-recaptcha-that-works - A reCAPTCHA bridge for React that works

React State Management

- redux - Predictable State Container for JavaScript Apps
- mobx - Simple, scalable state management
- react-query
- flux - Application architecture for building user interfaces
- recoil - Experimental state management library for React apps
- xstate-react - State machines and statecharts for the modern web
- zustand - Bear necessities for state management in React
- easy-peasy - Vegetarian friendly state for React
- hookstate - The simple but very powerful and incredibly fast state management for React that is based on hooks
- effector - Fast and powerful reactive state manager
- reactn - React, but with built-in global state management

React AR and VR

- Viro React - Platform for rapidly building AR/VR applications using React Native

React Renderers

- react-three-fiber - A React renderer for Three.js
- react-pdf - Create PDF files using React
- ink - React for interactive command-line apps
- react-blessed - A React renderer for blessed terminal interface library
- react-sketchapp - Render React components to Sketch
- react-figma - A React renderer for Figma
- react-nil - A react null renderer
- remotion - Create videos programmatically in React

Forms

- formik - Build forms in React, without the tears
- react-hook-form - React Hooks for forms validation

- react-jsonschema-form - A React component for building Web forms from JSON Schema
- react-final-form - High performance subscription-based form state management for React
- unform - Performance-focused API for React forms
- formily - Alibaba Group Unified Form Solution
- uniforms - A React library for building forms from any schema
- formsy-react - A form input builder and validator for React
- react-formal - Sophisticated HTML form management for React

Autocomplete

- react-autocomplete by @rackt - WAI-ARIA compliant React autocomplete $Archived, read-only$
- react-autosuggest by @moroshko - WAI-ARIA compliant React autosuggest component
- react-autocomplete by @eliseumds - Just tasting some ReactJS + RxJS
- react-autocomplete by @prometheusresearch - Autocomplete widget based on React
- instatype by @gragland - Simple react autocomplete component
- downshift - 🏎 Primitives to build simple, flexible, WAI-ARIA compliant enhanced input React components
- React Bootstrap Typeahead - A React-based typeahead that relies on Bootstrap for styling and was originally inspired by Twitter's typeahead.js.

Graphics

- react-art - React Bridge to the ART Drawing Library
- react-canvas - High performance `<canvas>` rendering for React components
- react-famous - Complex 3D animations UI at 60 FPS with Famo.us
- react-kinetic - HTML5 Canvas via KineticJS using React
- react-svg-morph - morph your svg components one into another
- react-hooks-svgdrawing - SVG Drawing with React hooks
- react-svg-pan-zoom - A React component that adds pan and zoom features to SVG.

Data Managing

- immer - Create the next immutable state by mutating the current one
- ReSub - A library for writing better React components and data stores
- immutable-js - Immutable Data Collections for Javascript
- baobab - JavaScript & TypeScript persistent and optionally immutable data tree with cursors
- WatermelonDB - 🍉 Reactive & asynchronous database for powerful React and React Native apps ⚡
- RxDB - A realtime Database for JavaScript Applications

Maps

- react-googlemaps - React interface to Google maps
- react-maps - A map component for React
- react-google-maps - React.js Google Maps integration component
- react-gmaps - A Google Maps component for React.js
- react-map-gl - A React wrapper for MapboxGL-js plus overlay API
- google-map-react - Isomorphic google map React component
- react-mapbox-gl - A mapbox-gl-js wrapper to make the API react friendly
- google-maps-react - A declarative Google Map React component using React, lazy-loading dependencies, current-location finder and a test-driven approach by the Fullstack React team.
- react-leaflet - React components for Leaflet maps
- react-geo - A set of geo-related components using react, antd, and ol
- pigeon-maps - ReactJS maps without external dependencies

Charts

- vx - Visualization components
- victory - A collection of composable React components for building interactive data visualizations
- react-vis - Data Visualization Components
- recharts - Redefined chart library built with React and D3
- nivo - Provides a rich set of data visualization components, built on top of the D3 and React libraries
- echarts-for-react - Apache ECharts components for React wrapper
- react-apexcharts - React Component for ApexCharts
- chartify - React plugin for building charts using CSS

React Native

> Framework for building native apps using React

React Native General Resources

- React Native Official Site
- React Native GitHub
- React Native Newsletter
- React Native Playground
- React Native Awesome List
- React Native StackOverflow
- React Native Radio

React Native Tutorials

- React Native Tutorial
- Introducing React Native: Building Apps with JavaScript
- Introduction to React Native: Building iOS Apps with JavaScript
- React Native Meets Async Functions
- Digital Smart Mirror lab with React Native
- The Beauty Of React Native: Building Your First iOS App With JavaScript $Part1$
- The Beauty Of React Native: Building Your First iOS App With JavaScript $Part2$
- A Mini-Course on React Native Flexbox
- A Complete Guide to Flexbox
- Test driving react native applications
- Using React Native With TypeScript

React Native Development Tools

- react-native-code-push - React Native module for CodePush

React Native Sample Apps

- HackerNews
- Ziliun
- FinanceReactNative
- SplashWalls
- NBAreact
- Bus Timetable

React Native Boilerplates

- Create React Native App - Create React Native apps that run on iOS, Android, and web
- Ignite - The hottest CLI for React Native, boilerplates, plugins, generators, and more!

React Native Awesome Components

- Expo - The Expo platform for making cross-platform mobile apps
- react-navigation - Routing and navigation for your React Native apps
- react-native-social-share - Use the iOS and Android native Twitter and Facebook share popup with React Native
- react-native-fbsdk - A wrapper around the iOS Facebook SDK
- react-native-side-menu - Simple customizable component to create side menu
- react-native-mapbox-gl - A Mapbox GL react native module
- react-native-icons - Quick and easy icons in React Native
- react-native-vector-icons - 3000 Customizable Icons for React Native with support for NavBar/TabBar

- react-native-google-signin - Google Signin for React Native
- react-native-picker-modal-view
- react-native-gifted-chat - The most complete chat UI for React Native
- react-native-fast-image - FastImage, performant React Native image component
- recyclerlistview - High performance listview for React Native and web!
- react-native-largelist - The best large list component for React Native
- react-native-gesture-handler - Declarative API exposing platform native touch and gesture system to React Native
- rn-placeholder - Display some placeholder stuff before rendering your text or media content in React Native

React Native Libraries

- sentry-react-native - Real-time crash reporting for your web apps, mobile apps, and games.
- realm-js - Realm is a mobile database: an alternative to SQLite & key-value stores
- react-native-device-info - Device Information for React Native iOS and Android
- react-native-react-bridge - A toolset to run React web app in React Native and handle communication between them.
- uncompress-react-native - Simple library to decompress files .zip, .rar, .cbz, .cbr in React Native.

Redux

> Predictable State Container for JavaScript Apps

Redux General Resources

- Redux GitHub
- Redux Official Site
- Awesome Redux List

Redux Tools

- react-redux - Official React bindings for Redux
- redux-toolkit - The official, opinionated, batteries-included toolset for efficient Redux development
- redux-devtools - DevTools for Redux with hot reloading, action replay, and customizable UI
- reselect - Selector library for Redux
- redux-thunk - Thunk middleware for redux
- redux-saga - An alternative side effect model for Redux apps
- connected-react-router - A Redux binding for React Router
- redux-form - A Higher Order Component using react-redux to keep form state
- normalizr - Normalizes nested JSON according to a schema
- redux-observable - RxJS middleware for Redux

- redux-undo - Higher order reducer to add undo/redo functionality to redux state containers
- redux-persist - Persist and rehydrate a redux store

Redux Tutorials

- Redux Essentials
- Redux Fundamentals
- Fundamentals of Redux Course from Dan Abramov
- Building React Applications with Idiomatic Redux

GraphQL

> A query language for your API

GraphQL General Resources

- GraphQL Official Site
- GraphQL Specification
- GraphQL Specification Repository

GraphQL Tools

- graphql-js - A reference implementation of GraphQL for **JavaScript**
- express-graphql - Create a GraphQL HTTP server with **Express**
- Apollo - Industry-standard GraphQL implementation
- GraphQL Playground - GraphQL IDE for better development workflows

GraphQL Tutorials

- GraphQL Introduction
- How to Graphql - The Fullstack Tutorial for GraphQL

GraphQL Implementations

- graphql-ruby - **Ruby** implementation of GraphQL
- graphql-java - GraphQL **Java** implementation
- sangria - **Scala** GraphQL client and server library
- graphql-php - A **PHP** port of GraphQL reference implementation
- graphene - GraphQL framework for **Python**
- graphql-dotnet - GraphQL for **. NET**
- graphql-go - GraphQL for **Go**
- juniper - GraphQL server library for **Rust**

Database Integration

- Hasura - Instant GraphQL for all your data
- Prisma - Next-generation ORM
  for Node.js and TypeScript
- graphql-sequelize - GraphQL & Relay for MySQL & Postgres via Sequelize

Relay

> Data-Driven React Applications

Relay General Resources

- Relay Offical Site
- Relay GitHub

Relay Tutorials

- Official Relay Getting Started
- Relay for Visual Learners
- Getting Started with Relay
- Relay and Routing

Relay Tools

- graphql-relay-js - A library to help construct a graphql-js server supporting react-relay
- react-router-relay - Relay integration for React Router
- relay-local-schema - Use Relay without a GraphQL server
- relay-codemod - Codemod scripts based for on jsodeshift to update Relay APIs

Videos

reactjsvideos.com

Important Talks

- Pete Hunt: React: Rethinking best practices - JSConf EU 2013
- Pete Hunt: React: Rethinking Best Practices $updated$ - JSConf. Asia 2013
- Tom Occhino and Jordan Walke: JS Apps at Facebook - JSConfUS 2013
- React: CSS in JS
- Pete Hunt: Be Predictable, Not Correct - Mountain West JavaScript 2014
- Hacker Way: Rethinking Web App Development at Facebook
- Christopher Chedeau: Why does React Scale? - JSConf2014

- Christopher Chedeau: React's Architecture - OSCON 2014
- Pete Hunt: React RESTful UI Rendering - Strange Loop 2014
- Pete Hunt: How Instagram.com Works - OSCON 2014
- Bill Fisher and Jing Chen: React and Flux - NewCircle Training 2014
- Sebastian Markbage: Minimal API Surface Area - JSConf EU 2014
- Avik Chaudhuri: JavaScript Testing and Static Type Systems at Scale - Scale 2014
- React Native & Relay: Bringing Modern Web Techniques to Mobile - f8 2015)
- Citrusbyte Presents GraphQL: A Horizontal Platform with Nick Schrock
- Laney Kuenzel: Mutations and Subscriptions in Relay - JSConf 2015
- React Today and Tomorrow and 90% Cleaner React With Hooks - React Conf 2018
- React Conferences
- React Videos
- Awesome React Talks

React.js Conf 2015 Playlist

ReactEurope Conf 2015 Day 1 Playlist

ReactEurope Conf 2015 Day 2 Playlist

ReactRally Conf 2015 Playlist

React.js Conf 2016 Playlist

React Amsterdam 2016 Playlist

ReactEurope Conf 2016 Day 1 Playlist

ReactEurope Conf 2016 Day 2 Playlist

ReactRally Conf 2016 Playlist

React Conf 2017 Playlist

React.js Amsterdam 2018 Playlist

React Amsterdam 2019 Playlist

Video Tutorials

- Trying React Hooks for the first time with Dan Abramov

Demo React Apps

- hackernews-react-graphql - Hacker News clone rewritten with universal JavaScript, using React and GraphQL
- react-reduction - Free Admin Template Built with React and Bootstrap4
- reactjs-tmdb-app - Responsive React The Movie Database App
- react-shopping-cart - Simple ecommerce cart application built with React Redux

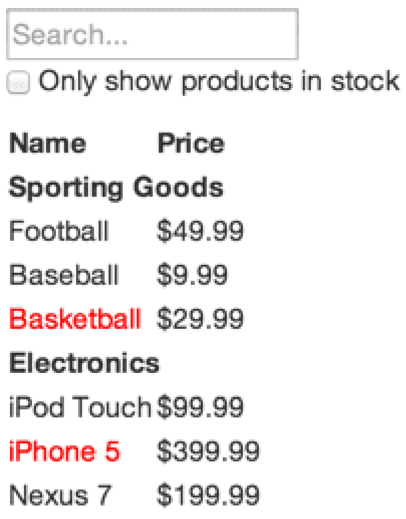
Real React Apps

- kibana - Your window into the Elastic Stack
- firefox debugger - The Firefox debugger that works anywhere
- spectrum – Simple, powerful online communities
- mattermost - Open source Slack alternative
- overreacted - Personal blog by Dan Abramov
- winamp2-js - Winamp 2 reimplemented for the browser
- dnote - A command line notebook with multi-device sync and web interface

Thinking in React

Start With A Mock

Imagine that we already have a JSON API and a mock from our designer. The mock looks like this:



Search...

Only show products in stock

Name	Price
Sporting Goods	
Football	$49.99
Baseball	$9.99
Basketball	$29.99
Electronics	
iPod Touch	$99.99
iPhone 5	$399.99
Nexus 7	$199.99

Our JSON API returns some data that looks like this:

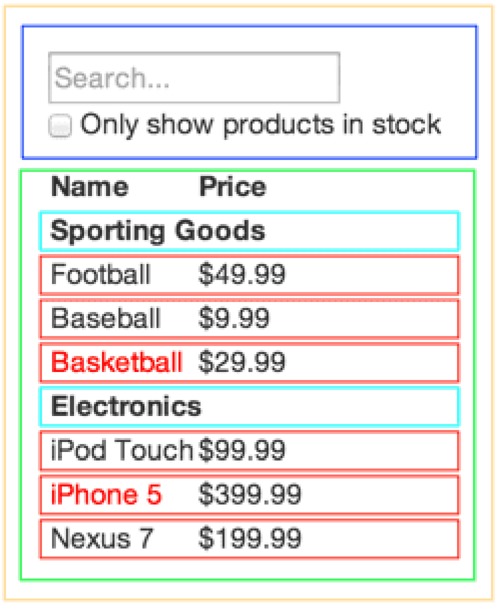
```
[
  {category: "Sporting Goods", price: "$49.99", stocked: true, name: "Football"},
  {category: "Sporting Goods", price: "$9.99", stocked: true, name: "Baseball"},
  {category: "Sporting Goods", price: "$29.99", stocked: false, name: "Basketball"},
  {category: "Electronics", price: "$99.99", stocked: true, name: "iPod Touch"},
  {category: "Electronics", price: "$399.99", stocked: false, name: "iPhone 5"},
  {category: "Electronics", price: "$199.99", stocked: true, name: "Nexus 7"}
];
```

Step 1: Break The UI Into A Component Hierarchy

The first thing you'll want to do is to draw boxes around every component $andsubcomponent$ in the mock and give them all names. If you're working with a designer, they may have already done this, so go talk to them! Their Photoshop layer names may end up being the names of your React components!

But how do you know what should be its own component? Use the same techniques for deciding if you should create a new function or object. One such technique is the single responsibility principle, that is, a component should ideally only do one thing. If it ends up growing, it should be decomposed into smaller subcomponents.

Since you're often displaying a JSON data model to a user, you'll find that if your model was built correctly, your UI $andtherefore your component structure$ will map nicely. That's because UI and data models tend to adhere to the same *information architecture*. Separate your UI into components, where each component matches one piece of your data model.



You’ll see here that we have five components in our app. We’ve italicized the data each component represents.

1. **`FilterableProductTable`** $orange$**:** contains the entirety of the example
2. **`SearchBar`** $blue$**:** receives all *user input*
3. **`ProductTable`** $green$**:** displays and filters the *data collection* based on *user input*
4. **`ProductCategoryRow`** $turquoise$**:** displays a heading for each *category*
5. **`ProductRow`** red**:** displays a row for each *product*

If you look at `ProductTable`, you’ll see that the table header $containingthe“Name”and“Price”labels$ isn’t its own component. This is a matter of preference, and there’s an argument to be made either way. For this example, we left it as part of `ProductTable` because it is part of rendering the *data collection* which is `ProductTable`’s responsibility. However, if this header grows to be complex $e.g., ifweweretoaddaffordancesforsorting$, it would certainly make sense to make this its own `ProductTableHeader` component.

Now that we’ve identified the components in our mock, let’s arrange them into a hierarchy. Components that appear within another component in the mock should appear as a child in the hierarchy:

- `FilterableProductTable`
    - `SearchBar`
    - `ProductTable`
        - `ProductCategoryRow`
        - `ProductRow`

Step 2: Build A Static Version in React

Now that you have your component hierarchy, it’s time to implement your app. The easiest way is to build a version that takes your data model and renders the UI but has no interactivity. It’s best to decouple these processes because building a static version requires a lot of typing and no thinking, and adding interactivity requires a lot of thinking and not a lot of typing. We’ll see why.

To build a static version of your app that renders your data model, you’ll want to build components that reuse other components and pass data using *props*. *props* are a way of passing data from parent to child. If you’re familiar with the concept of *state*, **don’t use state at all** to build this static version. State is reserved only for interactivity, that is, data that changes over time. Since this is a static version of the app, you don’t need it.

You can build top-down or bottom-up. That is, you can either start with building the components higher up in the hierarchy $i.e.startingwith$‘$FilterableProductTable$‘ or with the ones lower in it ‘$ProductRow$‘. In simpler examples, it’s usually easier to go top-down, and on larger projects, it’s easier to go bottom-up and write tests as you build.

At the end of this step, you’ll have a library of reusable components that render your data model. The components will only have `render()` methods since this is a static version of your app. The component at the top of the hierarchy ‘$FilterableProductTable$‘ will take your data model as a prop. If you make a change to your underlying data model and call `ReactDOM.render()` again, the UI will be updated. You can see how your UI is updated and where to make changes. React’s **one-way data flow** ParseError: KaTeX parse error: Expected group after '_' at position 29: …one-way binding__ keeps everything modular and fast.

Refer to the React docs if you need help executing this step.

A Brief Interlude: Props vs State

There are two types of “model” data in React: props and state. It’s important to understand the distinction between the two; skim the official React docs if you aren’t sure what the difference is. See also FAQ: What is the difference between state and props?

Step 3: Identify The Minimal $butcomplete$ Representation Of UI State

To make your UI interactive, you need to be able to trigger changes to your underlying data model. React achieves this with **state**.

To build your app correctly, you first need to think of the minimal set of mutable state that your app needs. The key here is DRY: *Don’t Repeat Yourself*. Figure out the absolute minimal representation of the state your application needs and compute everything else you need on-demand. For example, if you’re building a TODO list, keep an array of the TODO items around; don’t keep a separate state variable for the count. Instead, when you want to render the TODO count, take the length of the TODO items array.

Think of all of the pieces of data in our example application. We have:

- The original list of products
- The search text the user has entered
- The value of the checkbox
- The filtered list of products

Let's go through each one and figure out which one is state. Ask three questions about each piece of data:

1. Is it passed in from a parent via props? If so, it probably isn't state.
2. Does it remain unchanged over time? If so, it probably isn't state.
3. Can you compute it based on any other state or props in your component? If so, it isn't state.

The original list of products is passed in as props, so that's not state. The search text and the checkbox seem to be state since they change over time and can't be computed from anything. And finally, the filtered list of products isn't state because it can be computed by combining the original list of products with the search text and value of the checkbox.

So finally, our state is:

- The search text the user has entered
- The value of the checkbox

Step 4: Identify Where Your State Should Live

OK, so we've identified what the minimal set of app state is. Next, we need to identify which component mutates, or *owns*, this state.

Remember: React is all about one-way data flow down the component hierarchy. It may not be immediately clear which component should own what state. **This is often the most challenging part for newcomers to understand, ** so follow these steps to figure it out:

For each piece of state in your application:

- Identify every component that renders something based on that state.
- Find a common owner component $asinglecomponentaboveallthecomponentsthatneedthestateinthehierarchy.$
- Either the common owner or another component higher up in the hierarchy should own the state.
- If you can't find a component where it makes sense to own the state, create a new component solely for holding the state and add it somewhere in the hierarchy above the common owner component.

Let's run through this strategy for our application:

- `ProductTable` needs to filter the product list based on state and `SearchBar` needs to display the search text and checked state.
- The common owner component is `FilterableProductTable`.

- It conceptually makes sense for the filter text and checked value to live in `FilterableProductTable`

Cool, so we've decided that our state lives in `FilterableProductTable`. First, add an instance property `this.state = {filterText: '', inStockOnly: false}` to `FilterableProductTable`'s `constructor` to reflect the initial state of your application. Then, pass `filterText` and `inStockOnly` to `ProductTable` and `SearchBar` as a prop. Finally, use these props to filter the rows in `ProductTable` and set the values of the form fields in `SearchBar`.

You can start seeing how your application will behave: set `filterText` to `"ball"` and refresh your app. You'll see that the data table is updated correctly.

Step 5: Add Inverse Data Flow

So far, we've built an app that renders correctly as a function of props and state flowing down the hierarchy. Now it's time to support data flowing the other way: the form components deep in the hierarchy need to update the state in `FilterableProductTable`.

React makes this data flow explicit to help you understand how your program works, but it does require a little more typing than traditional two-way data binding.

If you try to type or check the box in the current version of the example, you'll see that React ignores your input. This is intentional, as we've set the `value` prop of the `input` to always be equal to the `state` passed in from `FilterableProductTable`.

Let's think about what we want to happen. We want to make sure that whenever the user changes the form, we update the state to reflect the user input. Since components should only update their own state, `FilterableProductTable` will pass callbacks to `SearchBar` that will fire whenever the state should be updated. We can use the `onChange` event on the inputs to be notified of it. The callbacks passed by `FilterableProductTable` will call `setState()`, and the app will be updated.

And That's It

Hopefully, this gives you an idea of how to think about building components and applications with React. While it may be a little more typing than you're used to, remember that code is read far more than it's written, and it's less difficult to read this modular, explicit code. As you start to build large libraries of components, you'll appreciate this explicitness and modularity, and with code reuse, your lines of code will start to shrink. 😃

Glossary

Single-page Application

A single-page application is an application that loads a single HTML page and all the necessary assets $suchasJavaScriptandCSS$ required for the application to run. Any interactions with the page or subsequent pages do not require a round trip to the server which means the page is not reloaded.

Though you may build a single-page application in React, it is not a requirement. React can also be used for enhancing small parts of existing websites with additional interactivity. Code written in React can coexist peacefully with markup rendered on the server by something like PHP, or with other client-side libraries. In fact, this is exactly how React is being used at Facebook.

ES6, ES2015, ES2016, etc

These acronyms all refer to the most recent versions of the ECMAScript Language Specification standard, which the JavaScript language is an implementation of. The ES6 version $alsoknownasES2015$ includes many additions to the previous versions such as: arrow functions, classes, template literals, `let` and `const` statements. You can learn more about specific versions here.

Compilers

A JavaScript compiler takes JavaScript code, transforms it and returns JavaScript code in a different format. The most common use case is to take ES6 syntax and transform it into syntax that older browsers are capable of interpreting. Babel is the compiler most commonly used with React.

Bundlers

Bundlers take JavaScript and CSS code written as separate modules $oftenhundredsofthem$, and combine them together into a few files better optimized for the browsers. Some bundlers commonly used in React applications include Webpack and Browserify.

Package Managers

Package managers are tools that allow you to manage dependencies in your project. npm and Yarn are two package managers commonly used in React applications. Both of them are clients for the same npm package registry.

CDN

CDN stands for Content Delivery Network. CDNs deliver cached, static content from a network of servers across the globe.

JSX

JSX is a syntax extension to JavaScript. It is similar to a template language, but it has full power of JavaScript. JSX gets compiled to `React.createElement()` calls which return plain JavaScript objects called "React elements". To get a basic introduction to JSX see the docs here and find a more in-depth tutorial on JSX here.

React DOM uses camelCase property naming convention instead of HTML attribute names. For example, `tabindex` becomes `tabIndex` in JSX. The attribute `class` is also written as `className` since `class` is a reserved word in JavaScript:

```
const name = 'Clementine';
ReactDOM.render(
  <h1 className="hello">My name is {name}!</h1>,
  document.getElementById('root')
);
```

Elements

React elements are the building blocks of React applications. One might confuse elements with a more widely known concept of "components". An element describes what you want to see on the screen. React elements are immutable.

```
const element = <h1>Hello, world</h1>;
```

Typically, elements are not used directly, but get returned from components.

Components

React components are small, reusable pieces of code that return a React element to be rendered to the page. The simplest version of React component is a plain JavaScript function that returns a React element:

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}
```

Components can also be ES6 classes:

```
class Welcome extends React.Component {
  render() {
    return <h1>Hello, {this.props.name}</h1>;
  }
}
```

Components can be broken down into distinct pieces of functionality and used within other components. Components can return other components, arrays, strings and numbers. A good rule of thumb is that if a part of your UI is used several times $Button, Panel, Avatar$, or is complex enough on its own $App, FeedStory, Comment$, it is a good candidate to be a reusable component. Component names should also always start with a capital letter $` < Wrapper/ > ` * *not * *` < wrapper/ > `$. See this documentation for more information on rendering components.

`props`

`props` are inputs to a React component. They are data passed down from a parent component to a child component.

Remember that `props` are readonly. They should not be modified in any way:

If you need to modify some value in response to user input or a network response, use `state` instead.

`props.children`

`props.children` is available on every component. It contains the content between the opening and closing tags of a component. For example:

```
<Welcome>Hello world!</Welcome>
```

The string `Hello world!` is available in `props.children` in the `Welcome` component:

```
function Welcome(props) {
  return <p>{props.children}</p>;
}
```

For components defined as classes, use `this.props.children` :

```
class Welcome extends React.Component {
  render() {
    return <p>{this.props.children}</p>;
  }
}
```

`state`

A component needs `state` when some data associated with it changes over time. For example, a `Checkbox` component might need `isChecked` in its state, and a `NewsFeed` component might want to keep track of `fetchedPosts` in its state.

The most important difference between `state` and `props` is that `props` are passed from a parent component, but `state` is managed by the component itself. A component cannot change its `props` , but it can change its `state` .

For each particular piece of changing data, there should be just one component that “owns” it in its state. Don’t try to synchronize states of two different components. Instead, lift it up to their closest shared ancestor, and pass it down as props to both of them.

Lifecycle Methods

Lifecycle methods are custom functionality that gets executed during the different phases of a component. There are methods available when the component gets created and inserted into the DOM ParseError: KaTeX parse error: Expected 'EOF', got '#' at position 57: …-component.html#̲mounting), when the component updates, and when the component gets unmounted or removed from the DOM.

Controlled vs. Uncontrolled Components

React has two different approaches to dealing with form inputs.

An input form element whose value is controlled by React is called a *controlled component*. When a user enters data into a controlled component a change event handler is triggered and your code decides whether the input is valid $byre - renderingwiththeupdatedvalue$. If you do not re-render then the form element will remain unchanged.

An *uncontrolled component* works like form elements do outside of React. When a user inputs data into a form field $aninputbox, dropdown, etc$ the updated information is reflected without React needing to do anything. However, this also means that you can't force the field to have a certain value.

In most cases you should use controlled components.

Keys

A "key" is a special string attribute you need to include when creating arrays of elements. Keys help React identify which items have changed, are added, or are removed. Keys should be given to the elements inside an array to give the elements a stable identity.

Keys only need to be unique among sibling elements in the same array. They don't need to be unique across the whole application or even a single component.

Don't pass something like `Math.random()` to keys. It is important that keys have a "stable identity" across re-renders so that React can determine when items are added, removed, or re-ordered. Ideally, keys should correspond to unique and stable identifiers coming from your data, such as `post.id` .

Refs

React supports a special attribute that you can attach to any component. The `ref` attribute can be an object created by `React.createRef()` function or a callback function, or a string $inlegacyAPI$. When the `ref` attribute is a callback function, the function receives the underlying DOM element or class instance $dependingonthetypeofelement$ as its argument. This allows you to have direct access to the DOM element or component instance.

Use refs sparingly. If you find yourself often using refs to "make things happen" in your app, consider getting more familiar with top-down data flow.

Events

Handling events with React elements has some syntactic differences:

- React event handlers are named using camelCase, rather than lowercase.
- With JSX you pass a function as the event handler, rather than a string.

Reconciliation

When a component's props or state change, React decides whether an actual DOM update is necessary by comparing the newly returned element with the previously rendered one. When they are not equal, React will update the DOM. This process is called "reconciliation".

Using Web Components in React

React and Web Components are built to solve different problems. Web Components provide strong encapsulation for reusable components, while React provides a declarative library that keeps the DOM in sync with your data. The two goals are complementary. As a developer, you are free to use React in your Web Components, or to use Web Components in React, or both.

Most people who use React don't use Web Components, but you may want to, especially if you are using third-party UI components that are written using Web Components.

Using Web Components in React

```
class HelloMessage extends React.Component {
  render() {
    return <div>Hello <x-search>{this.props.name}</x-search>!</div>;
  }
}
```

> Note:
>
> Web Components often expose an imperative API. For instance, a `video` Web Component might expose `play()` and `pause()` functions. To access the imperative APIs of a Web Component, you will need to use a ref to interact with the DOM node directly. If you are using third-party Web Components, the best solution is to write a React component that behaves as a wrapper for your Web Component.
>
> Events emitted by a Web Component may not properly propagate through a React render tree. You will need to manually attach event handlers to handle these events within your React components.

One common confusion is that Web Components use "class" instead of "className".

```
function BrickFlipbox() {
  return (
    <brick-flipbox class="demo">
      <div>front</div>
      <div>back</div>
    </brick-flipbox>
  );
}
```

Using React in your Web Components

```
class XSearch extends HTMLElement {
  connectedCallback() {
    const mountPoint = document.createElement('span');
    this.attachShadow({ mode: 'open' }).appendChild(mountPoint);

    const name = this.getAttribute('name');
    const url = 'https://www.google.com/search?q=' + encodeURIComponent(name);
    ReactDOM.render(<a href={url}>{name}</a>, mountPoint);
  }
}
customElements.define('x-search', XSearch);
```