#### Web App Front-End Development Best Practices 2017 - A Lab Book

If we complicate things, they get less simple. - Professor at Cambridge University

... Simplifications have had a much greater long-range scientific impact than individual feats of ingenuity. The opportunity for simplification is very encouraging, because in all examples that come to mind the simple and elegant systems tend to be easier and faster to design and get right, more efficient in execution, and much more reliable than the more contrived contraptions that have to be debugged into some degree of acceptability ... Simplicity and elegance are unpopular because they require hard work and discipline to achieve and education to be appreciated. - Edsger W. Dijkstra

Complexity is a sign of technical immaturity. Simplicity of use is the real sign of a well designed product whether it is an ATM or a Patriot missile. - Daniel T. Ling

Simplicity is the soul of efficiency. - Austin Freeman, 'The Eye of Osiris'

La perfection est atteinte non quand il ne reste rien à ajouter, mais quand il ne reste rien à enlever. [Perfection is not achieved when nothing needs to be added, but when there is nothing left to remove.] - Antoine de Saint-Exupéry

Controlling complexity is the essence of computer programming. - Brian Kernighan

Der Unterschied zwischen einem guten und einem schlechten Architekten besteht heute darin, daß dieser jeder Versuchung erliegt, während der rechte ihr standhält. [The difference between a good and a bad architect today is that one yields to every temptation, while the right one resists it.] - Ludwig Wittgenstein

Simplicity does not precede complexity, but follows it. - Alan J. Perlis

### Wolfgang Gehner - Vic Cekvenich

# Web App Front-End Development Best Practices 2017

### A Lab Book

Kindly supported by: appthings.io

1st Edition 2017

Copyright © Wolfgang Gehner and Vic Cekvenich

Contact sales@appthings.io to obtain rights to reproduce this book in any form.

## Preface

### Are you a mechanic or a driver?

### What you learn in this book

### Best Practices

### Real life

### Why read this book?

### What you should already know

### Intellectual Property and License

### About the Authors

### Contact Information

### Special Thanks

# Table of Contents

### Part I: The Basics

* [1: Hello World and Markdown](file:///C:/Users/Wolfgang/git/tutorial/docs/book/1-helloWorld/)
* [2: Material Design, SASS and Pug](file:///C:/Users/Wolfgang/git/tutorial/docs/book/2-theBasics/)
* [3: Go-Live - Deploy to Cloud and use a CDN](file:///C:/Users/Wolfgang/git/tutorial/docs/book/3-goLive/)
* [4: AppShell, Turbo and Caching for Performance](file:///C:/Users/Wolfgang/git/tutorial/docs/book/4-appShell/)

### Part II: The Bricks and Mortar

* [5: Accelerated Mobile Pages (AMP) and Node.js](file:///C:/Users/Wolfgang/git/tutorial/docs/book/5-amp/)
* [6: UI Reading from an API](file:///C:/Users/Wolfgang/git/tutorial/docs/book/6-read/)
* [7: UI Writing to an API (Example: Firebase)](file:///C:/Users/Wolfgang/git/tutorial/docs/book/7-write/)
* [8: Login Security and Tokens](file:///C:/Users/Wolfgang/git/tutorial/docs/book/8-security/)

### Part III: The Perspectives

* [9: Standard Web Components and their Communication](file:///C:/Users/Wolfgang/git/tutorial/docs/book/9-comps/)
* [10: Mobile App (Phonegap)](file:///C:/Users/Wolfgang/git/tutorial/docs/book/10-mobile/)
* 11: Next Edition: Extreme SEO with server-side rendering (SSR) of Web Components and server-side BLX

# Part I: The Basics

## Lab 1: Hello World - Using Markdown with Preprocessor

1. Downloand and unzip topseed-helloworld-master.zip from <https://github.com/topseed/topseed-helloworld> to your location of choice on your developer machine.
2. Open your Google Chrome web browser and install the 'Web Server for Chrome' app from [https://chrome.google.com/webstore/search/Web%20Server?\_category=apps](https://chrome.google.com/webstore/search/Web Server?_category=apps). Launch the app, click the 'Choose Folder' button and select the /helloworld-webroot folder under /topseed-helloworld/topseed-srv. Also ensure Options has 'Automatically show index.html' checked. Ensure the Web Server is STARTED, then navigate to the proposed URL (eg. [http://127.0.0.1:8887](http://127.0.0.1:8887/). You should see a demo website. Explore the site. It uses 'responsive design'. Resize the browser from fullscreen to narrow to see the layout adapt.
3. Download, install and run 'Visual Studio Code' (VS Code) from <https://code.visualstudio.com/download>. From 'File' menu, choose 'Open Folder...' and select folder /topseed-helloworld. When the project is loaded, inspect the default entry page at /topsseed-srv/helloworld-webroot/index.html We like VS Code, but you can use any other editor of your choice.
4. Download, install and run Prepros (Unlimited Trial) from <https://prepros.io/downloads>. Use the + button on the bottom left to add the folder /helloworld-webroot (under /topseed-srv) as a new project. In Project/Settings/Compiler Settings/Markdown, uncheck 'Wrap with Html'.
5. In the browser, return to the home page (e.g. [http://127.0.0.1:8887](http://127.0.0.1:8887/)). In VS Code, open /helloworld-webroot/page/one/\_hello.md and prefix the text with 'Hello World!' Save the file. In the same folder, inspect \_hello.html. Prepros will have 'preprocessed' your edited 'Markdown' file to HTML. Refresh the browser and see the edits. Search Google for 'Markdown syntax'.
6. Optional: Use Prepros to auto-refresh on edits. In Prepros, click on the arrows on the right side of the prepros project name 'helloworld-webroot', check Sync Browsers and click Preview. (Or rightclick on 'helloworld-webroot' and select 'Open Live Preview.') A browser should open and display the Home page. In VS Code, edit and save index.md again. The opened browser should refresh and include your changes.

## Lab 2: The Basics - Material Design, SASS and Pug

1. Ensure you have the sample project topseed-helloworld open in VS Code and Prepros running on the /topseed-webroot folder. Per Lab 1, make sure the home page is running in a browser.
2. Visit and browse these sites: [materialpalette.com](http://materialpalette.com/), [design.google.com](https://design.google.com/), [material.io](https://material.io/) and [material.io/guidelines/#](https://material.io/guidelines/). We follow Material design guidelines, put forward by Google, to make our sites and mobile apps look better.
3. Find /\_sass/main.css in the home page source, and inspect it. Other than fonts, all CSS used on the site is in this one CSS.
4. Visit [https://www.muicss.com](https://www.muicss.com/). Read 'Getting Started/Introduction', then browse the section 'CSS/JS'. We use the MUI CSS libraries in our project. Inspect /\_sass/\_base.scss in /helloworld-webroot. SCSS is CSS with some added features, such as $variables and @import. You can turn any CSS file into SCSS by just changing the ending to .scss.
5. Inspect /\_sass/\_colors.scss in /helloworld-webroot.See a color scheme we generated with materialpalette.com. Use /\* \*/ to comment it out. Go back to [http://materialpalette.com](http://materialpalette.com/), pick two colors, download your own palette in CSS format and paste it after the section you commented out. Save and refresh the browser, and you should see your new color scheme applied to the site. Revert to the palette you had commented out.
6. Inspect /\_sass/main.sass. SASS uses a special syntax without the curly braces or ';' at the end of a line that are used in CSS (and SCSS). SASS can also use SCSS imports such as @import '\_base'. Prepros compiles the .sass and its dependent .scss imports into the single 'main.css' used in the browser. In Prepros Files select \_main.sass, check 'Minify Css', and click Process File. Open /\_sass/main.css and see that it is now minified.
7. Inspect /page/two/\_buttons.html Copy the entire HTML. Go to [http://html2jade.org](http://html2jade.org/) (Pug used to be called Jade) and paste this html. In the right pane you see Pug markup, a way to write html without having to worry about closing tags. We write Pug.
8. Inspect the code snippet at /page/two/\_buttons.pug. It should match the output of html2jade. Every .pug file in the project has a corresponding .html used by the browser, with the exception of include's within another pug file. Inspect /page/two/index.pug to find the 'include \_buttons'. Inspect /page/two/index.html to find the generated buttons html.
9. Watch the video 'Do you Even JADE bro' at https://www.youtube.com/watch?v=wzAWI9h3q18. Once you know how to write Pug, you can generate beautiful html quickly.

## Lab 3: Go-Live - Deploy to Cloud and use a CDN

1. Go to [https://zeit.co](https://zeit.co/) and create a free account. We use their NOW product to deploy our apps for testing in the cloud. Install the NOW client. Instead, you could zip up the /helloworld-webroot folder and deploy to any static web hosting service via FTP. However, we will need some "dynamic", server-side features later, so we wrap our static webroot content with some code that works on a Node.js server, such as used by NOW. Rename /helloworld-webroot/cache.mfx to cache.mf.
2. To deploy, drag the /topseed-srv folder into the ZEIT.co NOW client. For advanced users: this folder includes index.js and package.json needed by Node.js. Once the deployment is completed, your clipboard will have a URL unique to the version you deployed. Test the URL in the browser. Bookmark the URL. You are in the Cloud and live on the Web!
3. The remainder of this lab is optional for development (but mandatory for QA/Staging/Production): To give your site a "proper" domain, you will need a domain name and DNS. If you do not have a domain yet, we recommend to register a cheap domain at [https://www.namecheap.com](https://www.namecheap.com/) now and have it use the namecheap DNS. If you already own a domain and host a site, e.g. at www.mydomain.com, you may want to configure a CNAME to map a 'staging' subdomain, such as staging.mydomain.com, so you can keep using www for your public site. See below for more detailed instructions.
4. For scalability and caching, you will also want to use a Content Delivery Network (CDN). With a CDN, you also get SSL/HTTPS for free. No need to buy an expensive SSL certificate. SSL is important when using advanced Javascript functions in the browser, such as cross-domain data requests. We recommend [https://www.cdn77.com](https://www.cdn77.com/). For this tutorial, register for the CDN77 14-day free trial now.
5. In the CDN77 web app, go to menu item CDN and click "ADD NEW CDN RESOURCE". Give it a label, such as 'staging.mydomain.com' and select "My Origin". As domain, specify HTTPS and the ZEIT.co DOMAIN from the URL you bookmarked under 2. (e.g. demos-oosnsyzlphl.now.sh). Click "CREATE CDN RESOURCE".
6. Choose 4-step setup with CNAME. Click "Add new CNAME", and "+ ADD CNAMES". Enter "staging.mydomain.com" and Click "ADD CNAME". Click "Go back to Integration". In Step 2, copy the DOMAIN NAME (AKA HOST), e.g 1234567890.rsc.cdn77.org, then follow instructions for your hosting provider. If your domain is with namecheap.com, do the following: On the namecheap dashbord, click "Manage" for your domain, and "Advanced DNS". Click "ADD NEW RECORD", select "CNAME" and enter the following: Host: staging Value: [DOMAIN NAME from clipboard, e.g. 1234567890.rsc.cdn77.org], TTL: Automatic. Click checkmark to save. No need to do CDN77 Step 3. One final step is to go to the "Other Settings" tab, check "HTTPS redirect" and click "SAVE CHANGES".
7. After an hour after the initial setup, you should be able to reach the deployed site in your browser under e.g. [https://staging.mydomain.com](https://staging.mydomain.com/). Note the use of "https". If you visit too quickly, the browser will complain that the site certificate is invalid. If this happens, try again after a while. The CDN caches static files for greater performance in multiple distributed datacenters. For advanced users: Cache-Control response headers are set in /topseed-srv/util/Decider.js.
8. Edit /helloworld-webroot/page/one/\_hello.md again (see Lab 1: 5.). To deploy the change, follow step 2 above. In CDN77 Overview, change "What is your domain?" to the new URL, and click "SAVE CHANGES". To make the changes appear on the CDN edge servers immediately, use CDN77 "CDN/Purge" on /page/one/.
9. Optional: Once you are ready to move from staging to production, you would either edit the CNAME for www to point to the same CDN domain (e.g. 1234567890.rsc.cdn77.org) or add a new CDN resource such as 'www.mydomain.com' that may also use a new ZEIT.co domain created when redeploying the app to ZEIT.co NOW (see 2.)

## Lab 4: AppShell, Turbo and Caching for Performance

1. Inspect /helloworld-webroot/page/one/index.pug and /page/two/index.pug. These Pug files show how the parts of the 'HTML they have in common are pulled from central places. Both pages use or 'extend' the TEMPLATE /\_part/\_baseShell.pug. Open this template file, and see that it has 'blocks' named 'head', 'main', and 'footer'. The Pug pages that extend this template define how to replace these blocks. For example, /page/one/index.pug defines that the head block consists of a page-specific title tag and an included fragment /\_part/\_header.pug. The same \_header.pug is re-used by /page/two/index.pug.
2. Inspect /helloworld-webroot/\_part/\_top.pug. This fragment represents the top menu and side drawer used on all pages. You can find it referenced in /\_part/\_baseShell.pug as 'include \_top'. Now inspect /page/one/index.html. This is the complete HTML which Prepros has collated together from template and fragments. Since the server has been configured to return the 'default' page index.html when the browser requests /page/one/, this is what the end user sees.
3. You have seen that a pug template or fragment can include other pug fragments. It can also include a fragment which is already in HTML format, such as /page/one/hello.html included in /page/one/index.html. hello.html is the result of processing the 'markdown' file hello.md. We find that Pug is not only a great way to write clean HTML, it is also very useful for defining layouts and reusing page elements.
4. To give the application a 'Single-Page Application' feel, we use an optional JavaScript library called topseed-turbo ('TT'). When a user navigates from one URL to the next, TT replaces only the part of the HTML body that is unique to each page (The section marked '#content-wrapper' in \_baseShell.pug and /\_js/main.js). TT is unobtrusively loaded by JavaScript included in the \_header.pug section (/setup-x.x.js). Using TT avoids 'page flash' and makes for smoother navigation, very similar to a native 'rich client' app where only individual panels are replaced on navigation. Smoother navigation increases the perceived performance of the application.
5. (Optional) Advanced JavaScript users may be interested to look at TS.onAppReady(UIinit) in /page/one/index.pug. TS stands for topseed-setup. TS receives an event when the #content-wrapper HTML is fully present in the browser (similar to a 'pageLoaded' event). When using TT, TS.onAppReady provides an important 'hook' for page-specific JavasSript that requires the DOM to be fully initialized.
6. Rename /helloworld-webroot/cache.mfx to /cache.mf and inspect it. The AppCache is a good way to improve performance of a web application. The file is referred to as 'manifest' in /\_part/\_baseshell.pug. If the manifest file is present, the browser will keep its listed resources in the browser cache 'forever', and serve them from there, without going to the network, until the cache.mf file itself changes. You can see what the browser does with this AppCache if you run a page in Developer mode (hit Ctrl-Shift+i and select the 'Console' tab in your Chrome browser). For development, we disable the AppCache by renaming it from .mf to .mfx. If you ever suspect the browser having cached an older version of a resource, rightclick the refresh button on the left of the URL in the Chrome browser and select "Empty Cache and Hard Reload".
7. In a later tutorial, we will discuss how we achieve additional performance gains by setting response headers to allow caching of content at the 'edge'.

# Part II: The Bricks and Mortar

## Lab 5: Accelerated Mobile Pages (AMP) and Node.js

1. AMP is a way to ensure that web pages render fast on mobile devices, for a better user experience (UX). Visit and browse [ampproject.org](https://www.ampproject.org/learn/overview/) to learn more about AMP. Other than fast rendering, one advantage of AMP pages is that Google will cache AMP pages for you for free. There are some restrictions as to what a valid AMP page can include (such as no JavaScript), but it allows you to use the full power of CSS. Download and unzip topseed-master.zip from [here](https://github.com/topseed/topseed) to your location of choice on your developer machine. We use the 'dynamic', server-side features of Node.js to help with serving AMP pages where appropriate. It is, however, possible to create a completely static site that serves AMP. In this tutorial, we show how both AMP and non-AMP versions can share common resources to limit content duplication.
2. Install Node.js and NPM. If you have a MAC, follow the instructions at http://blog.teamtreehouse.com/install-node-js-npm-mac. If you have Windows, download and run the Installer from <https://nodejs.org/en/download/>, accept the default settings, and RESTART your computer. Open the project you downloaded in step 2 above with VS Code. Select menu View-Integrated Terminal. On the command line that opens, change directories with 'cd topseed-srv'. To test Node, type 'node -v' and hit Enter. You should see a version number like v8.0.0. To test NPM, type 'npm -v' and hit Enter. You should see a version number like 4.0.3. Use Google to troubleshoot the install for your OS if necessary.
3. You are almost ready to run the topseed app in Node. First you will use NPM to download the 'dependencies' listed in /topseed-srv/package-json. Go back to the Terminal Shell, make sure you are in the /topseed-srv directory, then type 'npm install' and hit Enter. This will install the dependencies in a '/node\_modules' directory. This will take a while, but the console will tell you when done. (Repeat this when you add a dependency yourself).
4. Inspect /topseed-srv/index.js. This is the JavaScript file that starts the app, including an 'express' http server. Return to the Terminal Shell (still in the /topseed-srv directory) and type 'node index' (you can omit the .js) and hit Enter to start the app. You should see output like 'App listening on port 8091 and 8092. (To stop it, you would press Ctrl+C).
5. In a browser, go to [http://localhost:8091](http://localhost:8091/) and see /page/one/ come up. Rightclick on the page to 'Show HTML source'. You will note that the head tag has AMP-specific content. To compare, review the HTML source or the non-AMP version at localhost:8092. In production, we would likely map port 8091 to m.mydomain.com and port 8092 to www.mydomain.com. Behind the scenes, we have a Node module at /topseed-srv/util/Decider.js. The function of Decider is twofold: a) If a request of /page/one/ is on the first port, it will return the HTML from /page/one/indexA.html; if the request is on the second port, it will return the HTML from /page/one/index.html. b) It will attempt to return the alternate version if the requested one does not exist, no matter what port the request came on. You can change port numbers in /topseed-srv/config/ServerConfig.js (restart with 'node index').
6. Let us inspect how AMP pages are composed differently, but share some resources. Inspect '/page/one/indexA.pug'. Our convention is to post-fix AMP-specific resources with a capital A. See how /indexA.pug uses \_baseShellA.pug and \_headerA.pug, but uses the same \_footer.pug as non-AMP /index.pug. Likewise, both /index.pug and /indexA.pug share the use of \_hello.html. indexA.pug also has a required 'canonical' link that points to the non-AMP version of the resource. Also note that /indexA,pug does not have the 'script' element, since AMP does not permit custom JavaScript. AMP has some custom tags/components to help building AMP pages; see <https://www.ampproject.org/docs/reference/components>.For example, JavaScript IS allowed inside an amp-iframe (example at [https://m.appthings.io](https://m.appthings.io/)).
7. Inspect /\_part/\_baseShellA.pug. It includes \_topA.pug, which is a non-JavaScript version of \_top.pug that uses the #sidedrawer anchor combined with CSS to obtain the slide-out functionality without JavaScript. Also inspect /\_part/\_headerA.pug. It includes ../\_sass/mainA.css inside a style(amp-custom='') tag. This means that the CSS for AMP is served inline rather than from a separate request. mainA.sass has the same content as main.sass, but having a separate file for mainA allows us to compress or 'mininize' it (we use Prepros on the individual file) for inline use, since AMP files have a total size limitation.
8. Both AMP and non-AMP versions have the same responsive design features (they adapt to screen size). The AMP version should just be served faster, as optimized and privileged by Google caching. This may be especially important for the home page, where you do not want to let the user wait unnecessarily for the page to render. For custom behavior, you can force the non-AMP version by using the whole path /page/one/index.html in links or in the browser. We could make all 'a href' links in the app to go to non-AMP versions by adding /index.html to them. We usually do this to take advantage of the (JavaScript-based) Topseed Turbo feature that provides for a smoother Single-Page Application feel.
9. In summary, the app supports fast AMP responses but provides for non-AMP fallback versions where necessary. For example, a 'Contact Us' page that uses JavaScript to validate submissions would not have an AMP version. Decider.js on the server-side helps to automatically serve the available version.
10. You can use an AMP validator to ensure your AMP is valid; see [ttps://validator.ampproject.org](https://validator.ampproject.org/) Your markup has to be valid to be indexed and cached by Google. If your page is online, you can revalidate and submit it at <https://search.google.com/search-console/amp>.

## Lab 6: UI Reading from an API

1. For this and the next lab, we work on an 'Admin' module that allows listing (and adding to) a 'Link Blog', or list of links. In a browser, go to [http://localhost:8091](http://localhost:8091/) and click on the 'Admin' menu item to navigate to the 'Topseed Admin Console' (We will add login security in another lab). The Admin module has its own appshell and menu (see /\_part/admin/). To signal a full screen refresh to turbo, we added a '#' to the link that leads to the Admin home screen at [http://localhost:8091/admin/home/#](http://localhost:8091/admin/home/). The refresh ensures that the standard menu (which would otherwise remain 'cached' and re-displayed) is replaced by the Admin menu.
2. Click on the 'Linkblog' menu item at <http://localhost:8091/admin/linkblog/>to navigate to a list of Linkblog items. Rather than composing a screen in full on a server, modern web apps often take a HTML page to the browser first and then let the browser call an API to 'fill in' the data, using JavaScript. This way, the static elements of a page can be served by a fast CDN, and the dynamic/data parts can be obtained directly from the data source.
3. The Linkblog list is populated with data coming from a JSON response to an API call. By default, the Lab project is configured to obtain the API response from a file at /topseed-webroot/linkblog/dummy.json or <http://localhost:8091/linkblog/dummy.json> (which, in production, would be part of a CDN). However, in the next lab we will call a real, live database API that resides on a separate API server.
4. Now click the 'Add Item' button and note how the browser URL changed to <http://localhost:8091/admin/linkblog/detail.html>. This is a 'stable URL' for the the data entry form to add/write a new item. 'Stable URL' means that the page is not lost when you hit the browser refresh button, and that the page is 'bookmarkable', which is almost always a good thing. For now, click 'Cancel' to return to the list at <http://localhost:8091/admin/linkblog/>.
5. Inspect /topseed-webroot/admin/linkblog/index.pug. This file defines the composition of the Linkblog list screen. In the 'HTML' part of this file, the list is represented by the pug statement 'table#grid' (<table id="grid">). The 'script.' part of the file is responsible for causing the grid data to be loaded and rendered. The dot at the end of 'script.' tells Pug to not convert what follows to HTML.
6. In 'script.' we load external JavaScript that encapsulates logic, which keeps the page HTML clean and designer-friendly. In this example TS.loadOnAppReady loads /admin/linkblog/LinkblogBusiness.js, then calls the UIinit function in the page. This function triggers the 'list' function of LinkblogBusiness, which asynchronously loads data from the API, and - once the data has been received - renders the grid as well as the data in the grid. Depending on how much data is to be loaded, a more advanced version of LinkblogBusiness could have a separate function (e.g. init()) to first render the grid without data, and the list() function would only load and render the list items.
7. Open /topseed-webroot/admin/linkblog/LinkblogBusiness.js and scroll to the end of the file. See how LinkblogBusiness wraps a 'SimpleBusiness' object instance named 'sb' that is returned to the page after we added a LinkblogDao 'Data Access Object' instance. The page-specific custom functionality of SimpleBusiness is added close to the top of the file, beginning with 'var SimpleBusiness = BLX.extend({'. We keep common business functionality in a base 'class' named BLX.js ('BusinessLogiX'), and common data service functionality in BDS.js (BaseDataService). (Both are loaded by /\_js/admin.js, the admin module version of main.js.)
8. In LinkblogBusiness.js we use 'var SimpleBusiness = BLX.extend({' instead of the more recent 'class SimpleBusiness extends BLX {', because Internet Explorer does not support 'class'. However, Microsoft's 'Edge' browser supports it. We will use 'class' once IE has lost its remaining popularity; see /admin/linkblog/LinkblogBusiness2.js for an example. Note the resulting improvement in function signatures e.g. 'list()' vs ',list: function()'. If you can exclude the use of IE (such as for an internal webapp or a mobile app) we recommend you use the 'class' version. You can read more about the IE-compatible version at <http://johnresig.com/blog/simple-javascript-inheritance>. To use the 'class' version, in this example you would use TS.loadOnAppReady('/admin/linkblog/LinkblogBusiness2.js' in /admin/linkblog/index.pug and /detail.pug and change /\_js/admin.js to load BLX2.js and BDS2.js instead of BLX.js and BDS.js.
9. In LinkblogBusiness.js, inspect the code section following ', list: function(listId)'. The JavaScript in /admin/linkblog/index.pug triggers this function with 'sb.list('#grid')'. Brushing over the details of obtaining the data for now, this function receives a '\_listPromise' of data from the configured API. When data is returned, the function \_renderList enters '\_listPromise.then(function(values){', builds the grid with the received values (a JSON array of rows) and renders it in the page element with id 'grid'. There are many ways to render lists. In this example we use <https://dataTables.net/> to deliver advanced grid sorting, searching and paging features.
10. JavaScript Promises are a modern way to manage process flow. They are especially useful for handling asynchronous calls such as http requests to an API that may require error handling, such as connection errors. They replace classic asynchronous callbacks that are prone to Callback Hell; see <http://callbackhell.com/>. A lot of public APIs, such as Google Firebase, use or allow the use of Promises, and you can 'promisify' those that don't. Promises are important, learn how they work at <http://www.telerik.com/blogs/what-is-the-point-of-promises>. The post <https://stackoverflow.com/questions/22539815/arent-promises-just-callbacks> may also help.
11. The LinkblogDao Data Access Object is responsible for obtaining the raw list data. See 'urlSpec' at the top of LinkblogBusiness.js and how LinkblogDao extends BDS (which contains common data access functionality). Also see at the bottom of LinkblogBusiness.js how urlSpec is passed to LinkblogDao with 'sb.linkblogDao = new LinkblogDao(urlSpec)'. In this case, unlike SimpleBusiness, LinkblogDao has no added functionality, so we could write 'sb.linkblogDao = new BDS(urlSpec) instead.
12. Inspect '\_/js/BDS.js and its 'selectList' function. It calls a shared static \_get function which uses fetch\_ to call the urlSpec URL and returns a promise of the response content. 'Fetch' is a modern replacement for Ajax/XMLHttpRequest. Fetch uses Promises! Read more about Fetch at <https://davidwalsh.name/fetch>. At the end of the Lab we learn how an API server can handle this fetch request. In this Lab, the content of /topseed-webroot/linkblog/dummy.json or <http://localhost:8091/linkblog/dummy.json> is returned.

## Lab 7: UI Writing to an API (Example: Firebase)

1. In a browser, return to the Linkblog list at <http://localhost:8091/admin/linkblog/>, and click the 'Add Item' button to see <http://localhost:8091/admin/linkblog/detail.html>. Note that the Date field is prepopulated with today's date. In VS Code, inspect /topseed-webroot/admin/linkblog/detail.pug. This file defines the composition of the Linkblog Add Item screen. In the 'HTML' part of this file, the data entry form represented by the pug statement 'form#form1' (<form id="form1">). Note that it has 'onsubmit='return false', because we do not want the browser default behavior of posting to a form action URL.
2. Similar to the list page, in 'script.' we load /admin/linkblog/LinkblogBusiness.js, and then call the UIinit function in the page. UIinit triggers the 'detail()' function of LinkblogBusiness, which loads today's date into the form. UIinit also specifies to call the LinkblogBusiness save() function when the form is submitted (when the 'Save' button is clicked) instead of the default behavior, and pass any authentication cookie along. Since we are working with similar data as in the list page, and the two pages belong together, we have chosen to augment LinkblogBusiness.js with the detail() and save() functions rather than creating separate LinkblogListBusiness and LinkblogAddBusiness. For a different module we would create a separate XxxBusinness.
3. Reopen /topseed-webroot/admin/linkblog/LinkblogBusiness.js and locate the 'detail: function('. We use <https://momentjs.com/> for date handling and jquery.jsForm <https://github.com/corinis/jsForm> to populate the form with the date. The necessary libraries are loaded in /\_js/admin.js. You see the result of the call to detail() when rendering of the page has completed.
4. Fill some data in the form and click 'Save'. You should see an alert that saving is not enabled: we are not yet configured to use a database that allows saving. In LinkblogBusiness.js, locate the 'save: function('. Once we enable 'update' in the urlSpec, processing will continue. We obtain the 'formData' with jquery.jsForm('get'), and pass it to the linkblogDao.update function. Once this has returned successfully ('promise.then('), we redirects to the list page. BLX/sb has base functionality using turbo for the new page load.
5. Reopen BDS and look for the 'update: function('. It calls a shared static \_post function which uses fetch\_ to call the urlSpec update URL.
6. It is time to connect LinkblogDao/BDS to a live database. Go to the top of LinkblogBusiness.js, comment out the first urlSpec and comment in the urlSpec that goes to localhost:8081, does not use dummy.json and also has an update route, unlike the first urlSpec. We included a small API server implementation in the topseed project at topseed/bsrv that we will run at localhost:8081. The API server in turn will call a Google Firebase service that you will configure in the next step. Why do we not call the Firebase API directly from the browser? We do not want the browser to hold the authentication credentials for the database connection (more about user authentication in the next Lab).
7. Log into your Gmail/Google account at https://mail.google.com (Create one if you don't have one). Navigate to [https://console.firebase.google.com](https://console.firebase.google.com/), click 'Add Project', enter 'mydb1' as Project name when prompted and click 'Create Project'. Click the 'gear' icon on the left menu (enlarge browser window if necessary to see it) to access Project Settings and click the 'Service Accounts' tab. Copy the 'databaseURL' value from the Node.js Admin SDK configuration snippet and paste it into the topseed/bsrv/config/ApiConfig.js DB\_URL return value. The line in ApiConfig.js should look something like 'get DB\_URL() { return 'https://mydb1-7b77e.firebaseio.com' }' On the Google Project Settings Service Accounts tab, click 'Generate New Private Key' and 'Generate Key'. From the download prompt, save the file into the /topseed/bsrv/scode/route/ds/ folder. Open BaseFB.js, and replace 'serviceKey.json' with the filename. The line should look something like const 'serviceAccount = require("./mydb1-7b77e-firebase-adminsdk-8swi6-f46e592e60.json")' Verify that the path starts with './'.
8. You are ready to start the API server that uses this configuration. In VS Code, open another Terminal Shell (Ctrl+Shift+`), type 'cd bsrv [enter]', do the 'npm install [enter] ' and then 'node index [Enter]. You should see console output 'API server listening at http://localhost:8081'. On the VS Code terminal tab, use the dropdown go to the tab for topseed-srv, and start topseed-srv if not already running. You should see console output 'Web server listening at http://localhost:8091' (and 8092).
9. In a browser, go to or refresh the Linkblog list at <http://localhost:8091/admin/linkblog/>. The list should now say 'No data available in table', because LinkblogBusiness urlSpec points to the newly configured API server that has an 'empty' database. Click 'Add Item', enter some values and click 'Save'. Your item should appear in the list. Back in the Google Firebase Console at <https://console.firebase.google.com/>, for the project 'mydb1', navigate to 'Database' from the menu tree on the left. You should be able to drill down to see the newly created entry in a table named 'links11'. If you like, return to the Linkblog list at <http://localhost:8091/admin/linkblog/> in the browser and add a few more items with 'Add Item'.
10. Optional: Learn how the API server at /bsrv works. Inspect /bsrv/index.js. The line beginninng with 'server.use('/linkblog'' specifies that any requests to localhost:8081/linkblog are handled by /scode/route/LinkblogService.js. Inspect this file. The function beginning with 'router.post('/'' specifies in the line 'const \_promise = linkblog.update(row)' that a post request will call the update function in /ds/Linkblog.js. Inspect this file. Linkblog.js specifies the table name as 'links11', but also extends BaseFB. (Since Linkblog.js runs in the Node.js server and not in the browser, we can use 'extends' instead of .extends([). BaseFB has the common functionality to create a Firebase connection, query and update the database. For this, it uses the firebase-admin package imported as a dependency in /bsrv/package.json. To add a row, the BaseFB.js function 'update(row)' was used. Similarly, a 'get' request to localhost:8081/linkblog uses BaseFB.js 'selectList()' to obtain the list of linkblog items.
11. Perspective: For a complete Create-Read-Update-Delete (CRUD) implementation, we would add single-row query, update and delete capablity. To edit existing items, we would navigate to '/detail.html?id=x' and, in LinkblogBusiness detail(), use the 'id' URL parameter to obtain the existing data querying by a primary key column or unique index on the database. If you were to use a database service provider other than Google Firebase, you would create an implementation of BaseFB.js using the alternative provider's API, following the provider's examples for Node.js.

## Lab 8: Login Security and Tokens

1. Until now we had login security for the Admin module disabled. Inspect /admin/login/index.pug, and look for '//sb.redirect('/admin/home/'). Uncomment it by removing '//'. Ensure the API service is running. Return to the home page at <http://localhost:8091/>, and click the 'Admin' menu item to see the login prompt. Use 'demo':'demo' as username:password and click 'Login'. In VS Code, return to /admin/login/index.pug. See how we open the modal dialog, bind the form and register the form submit. Modal dialogs are natively supported by the Chrome browser. We use a 'polyfill' (loaded in /\_js/admin.js) for compatibility with other browsers (Edge and IE need a little bit of CSS love). LoginBusiness 'login()' contacts the API server (it could be a separate server dedicated to authentication). If the credentials match, LoginBusiness receives an 'OK' response, and we redirect to /admin/home/. If an authentication error is received, we display it in a div with id 'error' on the page.
2. Along with the 'OK' response, the API server returned a 'token' string. We store the token as a cookie (name: 'auth') in the browser. After the initial login, we use this token on all pages and for all API calls that require auhentication. While instead we could have stored username and password as a cookie, the advantage of using a token after initial authentication is that it can be heavily encoded (by the server). For maximum security and defense against network sniffing, the token could even be changed on every request.
3. To protecte the Linkblog pages with a login, open /admin/linkblog/index.pug and /detail.pug and uncomment //sb.ensureLogin(...). This function checks for the existence of the 'auth' cookie. If this cookie doesn't exist, it means that the user has not logged in, and he is redirected to the login screen. We destroy the 'auth' cookie on logout; see 'script.' in /admin/logout/index.pug. Logout is triggered when clicking the admin menu 'Logout' item.
4. There is one more thing to improve. In our flow, on full page refresh, the browser renders the static parts of the page before ensuring login. The ugly result is that the admin menu bar, static page content and footer may briefly display before a redirect to the login page happens. This is by design; we allow a CDN to cache the static (non-data) parts of admin module pages. However, we can apply a visual improvement to achieve better transitions. Open '\_sass/admin.sass' and uncomment the last two lines, beginning with '//body > \*'. This sets the opacity of the immediate children of the admin html body to a very low 0.1 by default (You can set it to 0, if you prefer). Linkblog.ensureLogin sets the opacity to 1 when the user has been confirmed to have successfully logged in, causing the page to display in full. Save admin.sass and try the login routine again to see the result. We could also imagine using a spinner during the transition; see <http://spin.js.org/spinner>.
5. For additional security, we send the token along when we make requests to the API server. If the API server is configured to require authentication, we ensure that the specific token received allows reading or writing the data before returning the data. If the token is not valid, we let the API service return a 403 'Forbidden' error. When LinkblogBusiness receives such an error, we show an alert and redirect to the login page. Open /bsrv/config/ApiConfig.js to configure which API calls are to be secured. In the line for 'get REQUIRE\_AUTH', uncomment 'write' so that it reads 'linkblog: ['write']'. Restart the server in the terminal window. This activates the token check in /bsrv/scode/route/LinkblogService.js in 'router.post('. The class TokenAuth contains our very simple Authentication functionalities: validate that username:password match demo:demo, and token always matches "abc".
6. You just activated token security for the Linkblog write (=save) function. Let's inspect how the token is passed to the API. In /admin/linkblog/detail.pug, with '$('#form1').submit({auth: Cookies.get('auth')}, sb.save)' we instruct to include the auth cookie as event data when calling LinkblogBusiness.save. In LinkblogBusiness 'save', we pass it on to the DAO in 'sb.linkblogDao.update(formData, e.data.auth)'. This continues through BDS.js update, \_post and fetch\_ functions, where the token is passed to the API as 'X-JToken' header. If configured as described above, in /bsrv/scode/route/LinkblogService.js the API reads this header and validates the token by calling '/bsrv/scode/route/ds/TokenAuth.js 'isTokenValidPromise'. Inspect this function. The returned promise throws an 'invalid token' error in the case of failure, which causes LinkblogService to abandon the update request and return a 403 error 'Forbidden'.
7. Reopen /admin/linkblog/LinkblogBusiness.js and find 'const \_updatePromise'. Here follows what happens in the UI as a result of the update attempt. If the update/save was successful, we redirect to the linkblog list page. If it failed because the token did not validate ('Forbidden'), show an alert and redirect to the login page. In case of other failures, simply show an alert. If you wish to simulate the errors, go to /admin/linkblog/detail.pug and make it send an invalid token with '$('#form1').submit({auth: 'xyz'}, sb.save)'. Attempt to add and save a new linkblog item. Revert to ' $('#form1').submit({auth: Cookies.get('auth')}, sb.save)' when done. Since our linkblog is expected to be public, we didn't implement authentication for the linkblogDAO.selectList API call, but this could easily be added.
8. This Lab demonstrated the fundamentals of the two principal authentication flows: login and token validation. A 'real life' authentication provider would likely be more advanced than TokenAuth.js. Tokens might need to be encoded and decoded. A production implementation might access a database of credentials, an in-memory database of valid tokens, or an asynchronously called external authentication service that may return its own promises for credential and token validations. In the shown implementation, sucessful login always redirects to the admin landing page. For production, we would probably enhance the login flow by keeping track of which protected page was requested and redirect to it after successful login.

# Part III: The Perspectives

## Lab 9: Standard Web Components and their Communication

1. The quest for the web equivalent of 'Legos' has been going on for a while. Legos are composable: you can connect pieces in different ways. They are reusable: the same kind of piece can be used in multiple places. Legos are also modular: you can connect one assembly to another one. Unlike Legos [(mostly; google kickstarter brixo )](https://www.kickstarter.com/projects/1068475467/brixo-building-blocks-meet-electricity-and-iot), however, components for the web also need to be able to communicate with each other and with the outside world. Finally, web components need to be insulated (in Lego terms, the color of one item must not 'bleed' into others). CSS 'bleed' has been a particular problem of older web component frameworks. To find an overall solution for great web components is not trivial, and many attempts have been made to achieve the holy grail of component-based development for the web.
2. In reality, component frameworks have had a short half-life. The last years has seen a rapid succession of candidates to be considered 'the best': Backbone, Knockout, Ember, Angular, Angular 2, React, Vue, Riot and Polymer. Find a comparison of component frameworks at <http://jeffcarp.github.io/frontend-hyperpolyglot/>. History shows that what you would pick today is likely not what you would pick 18 months from now. Assuming you don't want to start from scratch and rewrite everything every 18 months, that presents you with a great challenge of how to write applications that you can expect to be both maintainable and future-safe.
3. Our best practice to deal with the evolving landscape of web components is to write webapps that are structured in a 'component-framework-agnostic' way, where you can potentially keep existing components around, but develop new components using a newer component framework. We achieve this by avoiding the use of the two kinds of common component framework features that create unattractive 'lock-in': custom navigation mechanisms (routers) and framework-specific component communication implementations. We use the browser's great native navigation mechanisms (URL and browser history) as shown in Lab 4, and framework-agnostic component communication as implemented in the sample project shown below.
4. There is, however, light at the end of the tunnel: a W3C standard for web components appears to be forming. Read about Standard Web Components at <https://en.wikipedia.org/wiki/Web_Components>. The Chrome browser already supports Standard Web Components natively, so there is nothing extra to load, and the polyfills for the other browers are lean and fast. The most recent contender for 'the best' component framework, Polymer, is actually developed on top of this emerging standard, by the same people at Google in charge of the standard reference implementation. We use standard web components as the default component framework for this Lab. However, our architectural concepts can also be applied when using other frameworks. May we tempt you to write a sample integration for your favorite component framework and contribute it to topseed?
5. Among other things, newer component frameworks solve the problem of CSS bleeding by using the 'Shadow DOM' construct. To work with web components you should understand Shadow DOM; read a good introduction at <https://www.html5rocks.com/en/tutorials/webcomponents/shadowdom/>. We prefer it over the more recent introduction at <https://developers.google.com/web/fundamentals/getting-started/primers/shadowdom>. 'Templates' and 'HTML Imports' are other Standard Web Component features which will become clear with the examples in this Lab. Finally, there is a 'Custom Element' API to create new HTML tags. However, the implementation of this API uses 'class' which is not supported by Internet Explorer. We will use the Custom Element API once IE has lost its remaining popularity, or when we can rule out IE in a corporate app.
6. Download and unzip topseed-webcomps-master.zip from [https://github.com/topseed/topseed-webcomp](https://github.com/topseed/topseed-webcomps) to your location of choice on your developer machine. Open the project in VS Code. Add the project folder /topseed-webcomps Prepros, but but ensure that Auto-Compile of Pug (Jade) is deactivated. In this project, Node.js compiles pug files on the fly when responding to HTTP requests so we don't need Preprose to precompile them. See the function 'pugComp' in /demo-srv/util/Decider.js. Since in production we cache generated HTML responses in the CDN, there is practically no impact on production performance, but the development project is a lot cleaner.
7. In VS Code, open Terminal Shell (Ctrl+Shift+`), type 'cd demo-srv [Enter]', do the 'npm install [Enter]' and then 'node index [Enter]'. You should see console output 'Web server listening at http://localhost:9081'. In the browser, navigate to [http://localhost:9081](http://localhost:9081/), and visit Dashboard, List, Circle, Prelist and List menu items. When writing a component, we always advise to make things work outside of a component first. 'Prelist' is a non-component list page. In VS Code, inspect its /demo-srv/root/page/list-0/index.pug. Similar to the admin linkblog in tutorial 6, this page uses '/page/list/ListBusiness.js' to load a list from a JSON response promise. For data binding, ListBusiness.list() uses doT.js, a fast 'moustache-style' template library. The doT template is embedded in the page html as a script of type 'text/x-dot-template'. Inspect ListBusiness.js, beginning with 'var templateText' to see the JavaScript used to render the template with data and attach it to '#myList' in the page. You can read more about doT at <http://www.javascriptoo.com/dot-js>.
8. Inspect the component version of the list page at /demo-srv/root/page/list/index.pug. It has a custom element in the html named 'list-el'. Custom elements must include a '-' (dash) in the tag name. The doT template has disappeard. The page 'script.' loads ListBusiness.js and the List component definition from the 'HTML Import' /\_webComps/List.html, and then calls sb.compList() to render an instance of the component. In /page/list/ListBusiness.js, compare the function 'compList' with the function 'list' used by the non-component version. In this example, the 'compList' function obtains the component instance with 'document.querySelector' and calls its list(values) function. We always pass data to a component, instead of making the component load data. This keeps the component simpler and more manageable. As a result of using a component, both the page and ListBusiness are somewhat cleaner; 'JavaScript mess' is hidden inside the component.
9. Optional: Inspect the list component implementation at /demo-srv/root/\_webComps/List.pug. You will find the Standard Web Component 'template' tag that includes the node #myList previously seen in the non-component page, as well as the x-dot-template. The 'script.' section creates a HTMLElement prototype named 'ListEl', specifies to attach the insulated shadow DOM (using TW.attachShadow) when an actual component instance is created from the prototype ('createdCallback'), and registers the element prototype as a custom element with the DOM, using TW.registerCustomElement. Also see the ListEl.list function implementation which places the databinding result into the '#myList' node in the shadow DOM. (You can find the the 'TW.' helper library function implementations in /demo-srv/root/\_js/setup-3.4.js.)
10. As of the time of writing, due to imperfect polyfills, CSS still bleeds in both directions in Firefox and Edge. For us this is not catastrophic because we already avoid most CSS namespace collisions by using BEM syntax when naming our own component styles. Remaining potential issues with third party styles used inside components will disappear once all browsers natively support Standard Web Components. Read <https://csswizardry.com/2013/01/mindbemding-getting-your-head-round-bem-syntax/> to lean how to use BEM; it is a best practice even when not using components. See /\_webComp/circle.pug for an example of BEM inside a component. You can also attempt to insulate component CSS by using a scope div (see use of .bgauge-el in /\_webComp/gauge.pug) It is also possible to write components that bleed 'on purpose', by making components use actual DOM vs shadow DOM. You may choose to do this if you have a well-managed, globally applicable CSS regime (ideally using BEM throughout), and are not worried about 3rd party style bleeding. Or you can use SASS to bring global CSS styles into the component, analog to using mainA.css inline with AMP as shown in tutorial 5.
11. In a browser, go to the dashboard page at <http://localhost:9081/page/dashboard/>. Inspect the dashboard page at /demo-srv/root/page/dashboard/index.pug for an example of using multiple components in one page. Find the reused 'list-el', as well as 'circle-el' and 'gauge-el' in the HTML. Inspect 'script.' 'function UIinit'. Note that The circle and gauge components are loaded from absolute URLs. This means that components can easily be shared across different web projects. In this example, ListBusiness is also used as the component communication 'message bus' (The message bus features are found in BLX, the base class for ListBusiness.) 'sb.compInit' connects each component to the bus, as long as the component implements a function named 'init'. (The non-dashboard examples didn't call sb.compInit because their components did not send or receive messages.) Because ListBusiness was already present for loading list data, it was convenient to use it as message bus as well. if you prefer, you can use a separate message bus instead. Just use 'const \_blx = new BLX(null)' on the page instead and make calls to '\_blx.compInit('my-comp')'.
12. On the dashboard page at <http://localhost:9081/page/dashboard/>, click on one of the list links see how the circle and gauge display values change. Repeat for each list item. The list component has detected that it is enabled to communicate and has pushed hidden list data (values for circle and gauge) to the bus (rather than just opening a new tab). Because circle and gouge components are also registered with the bus, they receive the values and use their component-specific implementation to update the display. This way, components are very loosely coupled.
13. Optional: Learn about the component communication implementation. First inspect the function 'compInit' at /demo-srv/root/page/list/ListBusiness.js. It injects the bus into the component. Look at the 'ListEl.init' function in /demo-srv/root/\_webComps/List.pug. The component stores a reference to the bus. In the same file, look for the 'text/x-dot-template' and see how 'ListEl.nav' is triggered when the user clicks on the link. Inspect the function ListEl.nav. It sends the data to the bus using key 'mySelection', see 'blx.emit('mySelection'. In the Circle component implementation at /topseed-webcomps/\_webComp/circle.pug, find function 'CircleEl.init' (the equivalent of 'ListEl.init' for the Circle web compoment). In '\_blx.on('mySelection',..' it specifies to update the component display when the bus received a message with key 'mySelection'. In summary, the components are 'loosely coupled' because neither component requires the presence of other components.
14. We like creating and using components when it make us more productive, and the code becomes more maintainable. We decide this per use case. Be your own judge!

## Lab 10: Mobile App (Phonegap)

1. In this Lab we will customize and install an Android app that uses the Cordova In-App Browser to provide a rich UI. Download and unzip topseed-mobile-master.zip from <https://github.com/topseed/topseed-mobile> to your location of choice on your developer machine. Open the project in VS Code. Download, install and run the Phonegap Desktop App from <https://github.com/phonegap/phonegap-app-desktop/releases>. In Phonegap Desktop, add the '/topseed-mobile' folder as a Project, and click the '>' button. A message 'Server is running on ...' should come up. Click on the URL to open the App in a browser. Resize the browser to mobile phone format.
2. The app is configured to show a splash screen and then display content obtained from https://m.appthings.io. We will make it display the content you published to the CDN in Lab 3 instead. As you edit the project, the Phonegap Desktop App watches for changes in the /www folder of the project and attempts to refresh the browser view on each change. In /www/js/index.js, replace the URL following 'window.open' with the homepage URL of the helloworld app you deployed in Lab 3, i.e. your equivalent of 'https://staging.mydomain.com/page/one/index.html'. If you only deployed to the CDN or zeit.co, use your equivalent of 'https://1234567890.rsc.cdn77.org/page/one/index.html'. or 'https://demos-oosnsyzlphl.now.sh/page/one/index.html'. If you never deployed, you can use the published version of this tutorial at 'https://docs.topseed.io/tutorial/0-agenda/index.html'. We include '/index.html' in the path to ensure we deliver the turbo (non-AMP) version of the page.
3. To allow the in-app browser to navigate to the newly configured URL, in /config.xml <allow-navigation href="https://m.appthings.io/\*" />, replace 'm.appthings.io' with your equivalent from the previous step. The app should now display your site at the Phonegap Desktop App Server URL. It is a labor of love to replace the splash screen logo (at /www/css/index.css) and the icons used by the phone operating system (see /config.xml 'icon' values) with your own; we will skip over it here. Optional: change the values for the app 'name' and 'version' in /config.xml.
4. To install the customized app on an Android phone, first create a zip file of the contents of the project folder /topseed-mobile. Create an account and login at [https://build.phonegap.com](https://build.phonegap.com/). On the 'Apps' tab at <https://build.phonegap.com/apps>, click the '+ new app' button and upload the zip file (If your project is in Github, you can use the github clone URL instead). The site will take a few moments to build the app. Meanwhile, also log into [https://build.phonegap.com](https://build.phonegap.com/) on your mobile phone. Once the build has completed, on your phone download, install and run the version for your phone's operating system (here: Android). Follow the installation prompts. Done! For fun, close the app and reopen it by tapping on the app icon on your phone UI.
5. See <http://docs.phonegap.com/> for further information about phonegap, for instruction how to deploy to Apple iOS,and how your app can access phone-specific APIs, such as address book, location information etc if necessary. We would use a 'service worker' to support offline-browsing as needed.
6. In this Lab, we have used the Cordova In-App Browser to render app content from a CDN. As shown in Lab 6, the app can also obtain data securely from separate API servers. Thanks to Appshell, Turbo and Caching introduced in Lab 4, the app has a smooth 'single page application' feel. As a result, we were able to develop a rich mobile app without requiring special Android SDK/iOS development skills.