BuildIt
5-day weather forecast test

Paul A Oliver

September 2016

Project strategy

Objective

- Demonstrate approach to designing and implementing an application development project.
- Demonstrate coding skills and style.

Scope

- Limited-time exercise (rather than to build fully functional app).
- Defined by the client at: buildit /org-design/Recruitment/Exercises/js_engineer.md

Method/ Process

- Use familiar IDE, HTML/CSS frameworks and testing frameworks to reduce development time.
- Conduct desk-based research of existing weather apps - (due to time restrictions) focus on top 2 iPhone / Android apps, (identified by user ratings) and search of weather forecast web pages. Identify key layouts & data displays, core functionality + additional functionality, core graphic design elements.
- Research OpenWeatherMap to determine scope and complexity of available functionality.
- Design the UI and the product architecture.
- Build and test on local machine.
- Deploy to www.hamel.com/weather for live testing.
- Minify and package the app, distribute via github – keep it simple (use zipped archive).
- Write up final documentation.

Existing weather apps & pages

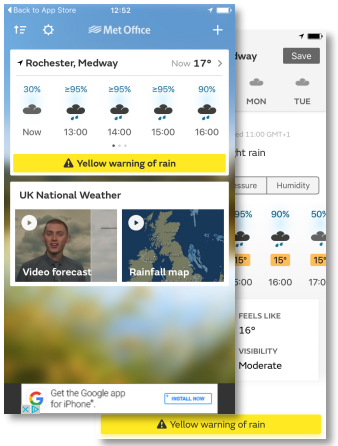
Mobile platforms

- Weather apps from the top providers share a common UI design (style and functionality) across mobile platforms.
- 2 main styles: 1) simple text, 2) visually dominant icons + text.
- Common variations:
  - Plain background / background image.
  - Current day only / current day + summary of multi-day forecast (on initial screen).
  - By hour / day only ‘summary’ of current day (on initial screen).
  - Focused information / ancillary information (on initial screen).

The Weather Channel

Yahoo

BBC

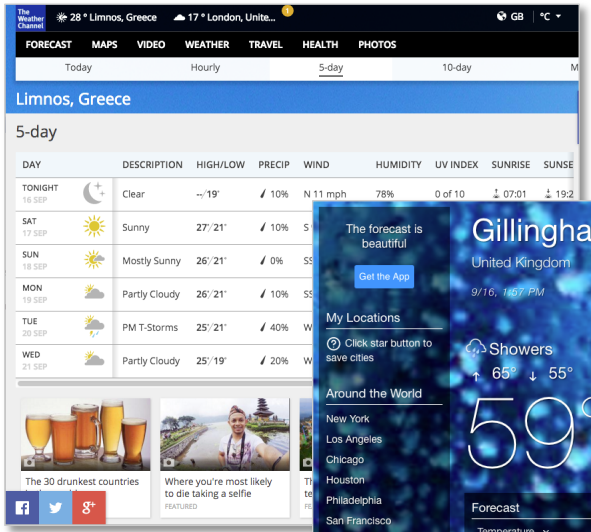


The Met Office (UK)

Existing weather apps & pages

Web sites

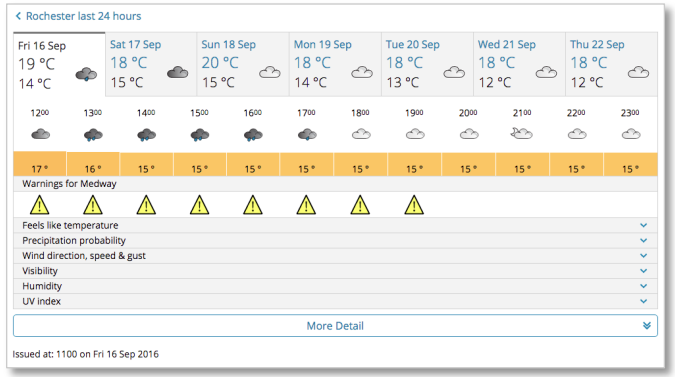
- Web page-based weather apps are a lot less sophisticated in their visual styling.
- Tend to show a lot of weather data on initial landing.
- Lots of ancillary information that clouds the main message.



The Weather Channel

Yahoo

BBC



The Met Office (UK)

Product strategy (1/2)

Objective

- Build single HTML5 page to display the 5 day weather forecast of a single city.
- Mimic style and functionality of existing weather apps – a 'familiar' look and feel will reduce barriers to adoption.
- Build ultra lightweight core functionality (get info and display) in discreet objects for unit testing.
- Build in space for later development of additional functionality.

Core functionality (version 1 – scope of this project)

- Fetch 5 day/ 3 hour weather forecast for fixed location (London, UK), from OpenWeatherMap.
- Use single language for UI (UK English) and Celsius for units.
- Limited displayed data (e.g. temp + description) for current day (only).
- Highlight of next 5 days.

Additional functionality (version 2+, delivered in stages, not to be implemented)

- Language support – user can change language. To simplify development, only use languages available from the OpenWeatherMap API.
- Location support – user can change location of forecast (free text typing + auto recognition of city by parsing restricted list of available cities).
- Refresh data periodically.

continued over

Product strategy (2/2)

Additional functionality (continued)

- Current location – identify current location using browser-based JavaScript* or call to third-party API. Set current location as default.
- Additional data – ability to show additional information, such as humidity and hourly temperatures.
- More detailed forecasts for days 2 – 5.
- Previous state and list of locations.
- User error messages.

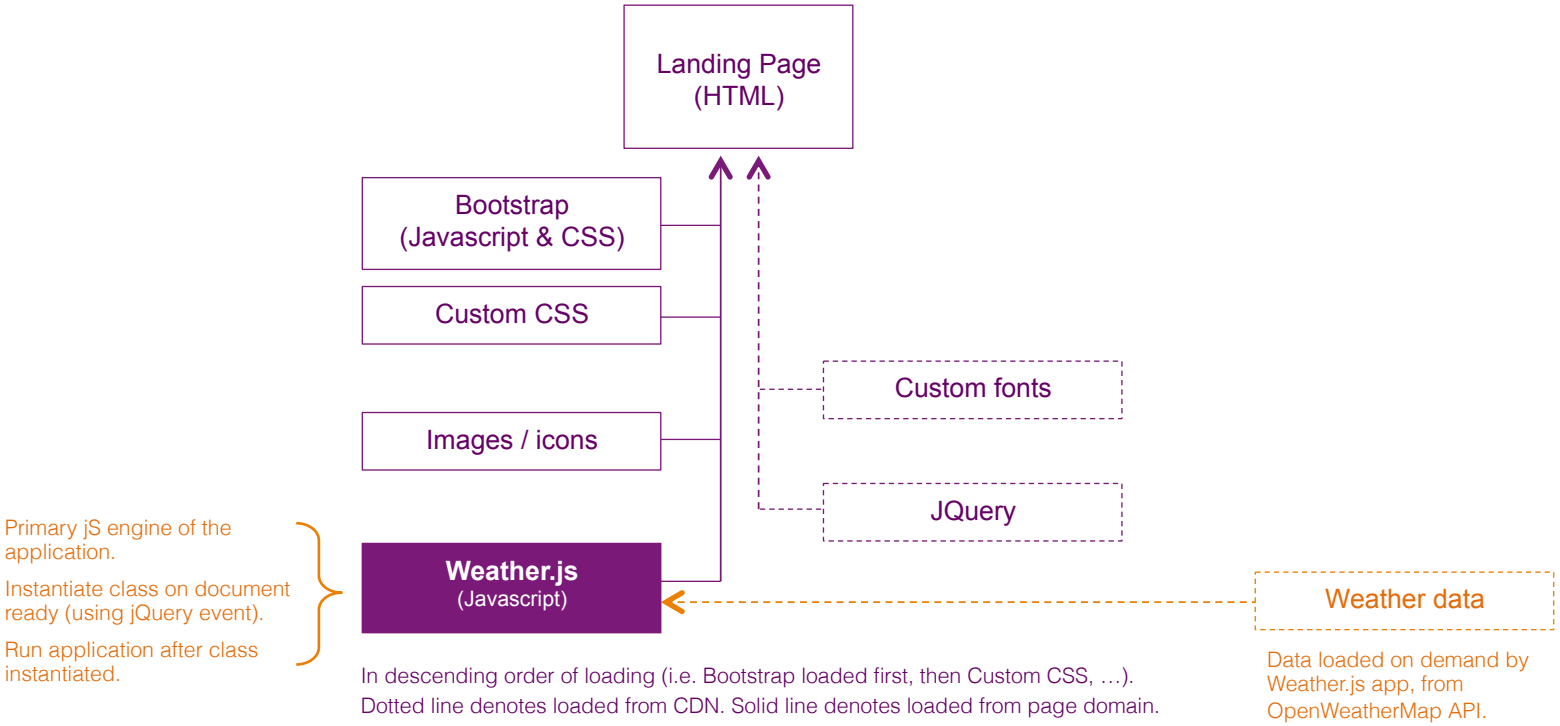
Technology stack

- HTML5 – single page, static code.
- Bootstrap framework – for cross platform display (probably not needed in version 1, but has very low implementation cost and will dramatically reduce development time in version 2).
- Custom font(s)**
- jQuery** (for custom classes and for Bootstrap).
- OO Javascript (core app).
- Zend IDE (development) + Jasmine testing framework.
- Static custom CSS (not much required!).

* Requires HTTPS for Chrome.
** For speed of development, access via CDN (3rd party dependency risk not an issue as already dependent upon 3rd-party OpenWeatherMap.

System architecture

Application



Weather.js: architecture features & build schedule

Architecture

Data API

- Build core engine to handle different APIs with minimal refactoring of the code (i.e. don't hard-wire OpenSourceMap and the specific 5 day/3 hour service) – reduces 3rd party risk and time to refactor code if required (as expense of slightly higher initial coding time).

Temperature values

- Display temperatures as integers.
- Round average temperature to nearest integer, round max temperature up, and min temperature down.

Data display (version 1)

- Display 3 hour forecast for today.
- Display high and low temps for following 6 days.
- Display short weather description.
- Clear display if not have valid value.
- Use service provider graphics – reduce build time.

Build schedule

Step 1 (initialisation)

- Class instantiation
- Version control
- Set up of primary data containers
- Setting of initial data values

Step 2 (data acquisition & processing)

- Forecast data accrual
- API call to OpenWeatherMap
- Data processing of API response.

Step 3 (data display)

- Display forecast (DOM manipulation).

Weather.js tests

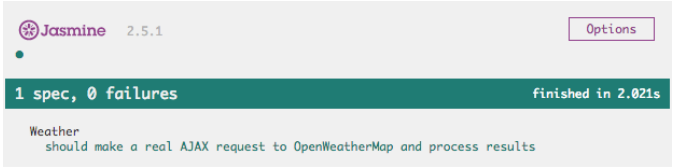
Step 1: Initialisation

jasmine files: SpecRunner.html, weather_initialise_spec.js

tests

- initialisation of the app
- initial settings.

Step 2: Data acquisition & processing



jasmine file: SpecRunner.html, weather_apiCall_spec.js

tests

- call was made to OpenWeatherMap API
- API response was processed
- API response was an object containing data, in the expected format.

Note

- Long execution time (> 2 seconds), due to the need to test after a delay to allow the API call to be made and completed. A setTimeout() call was made in the test to accommodate the asynchronous AJAX call in the app.

Step 3: Display

jasmine file: SpecRunnerDisplay.html, weather_idisplay_spec.js

tests

- DOM rendering of the weather data.

Finished product