Before start

Lab 1-2 Unix commands, regular expression (working with Linux)

Lab 3 Git (code management, for project)

Lab 4 Agile (software development workflow)

Lab 5 SQL (mysql, database basic)

Lab 6-7 web development (REST, heroku)

Lab 6 - REST web services

Objectives

- Start a local web server
- Create a web page and display on a web server
- Use jQuery’s Ajax to get data from a REST API web server
- Parse JSON data and display on a web page
- Please pair program today

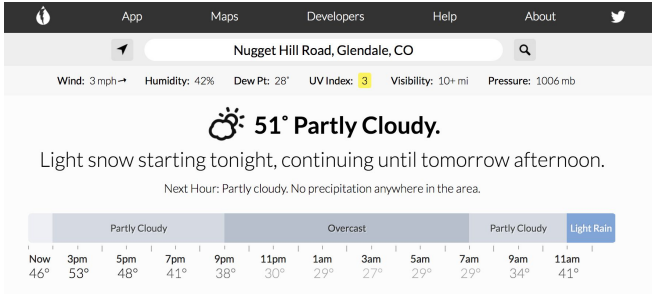
Pre-lab

1. Ensure you have a **web server** installed
    a. Install **python**
    b. Running a simple HTTP server in current directory: **python -m SimpleHTTPServer 8000**

2. Install the **jq** command line tool for formatting JSON data
    a. Linux: **apt-get install jq**
    b. OS X: **brew install jq**

Pre-lab

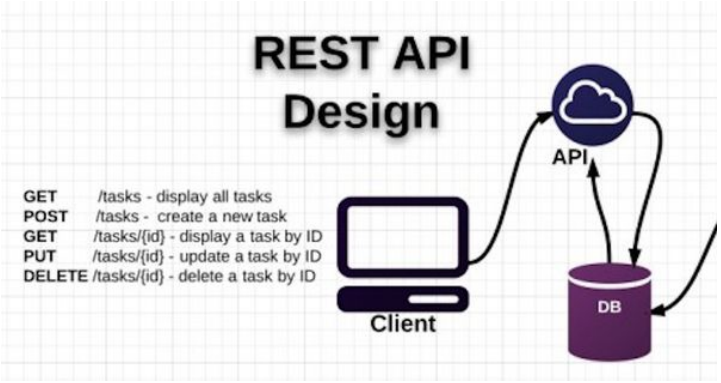
3. Sign up for **Dark Sky Forecast API**



API: Application Programming Interface

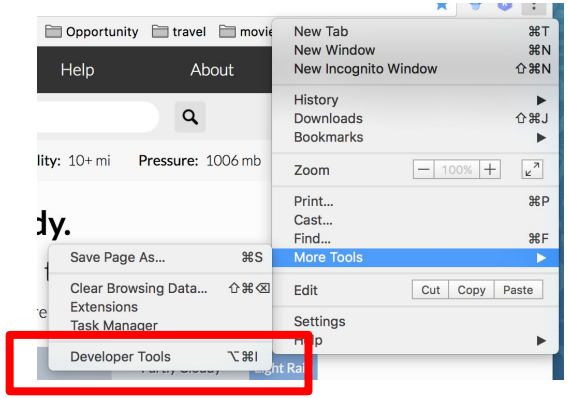
Without API:
An app finds the current weather in London by opening http://www.weather.com/ and reading the webpage like a human does, interpreting the content.

With API:
An app finds the current weather in London by sending a message to the **weather.com API** (in a structured format like JSON). The weather.com API then replies with a structured response.

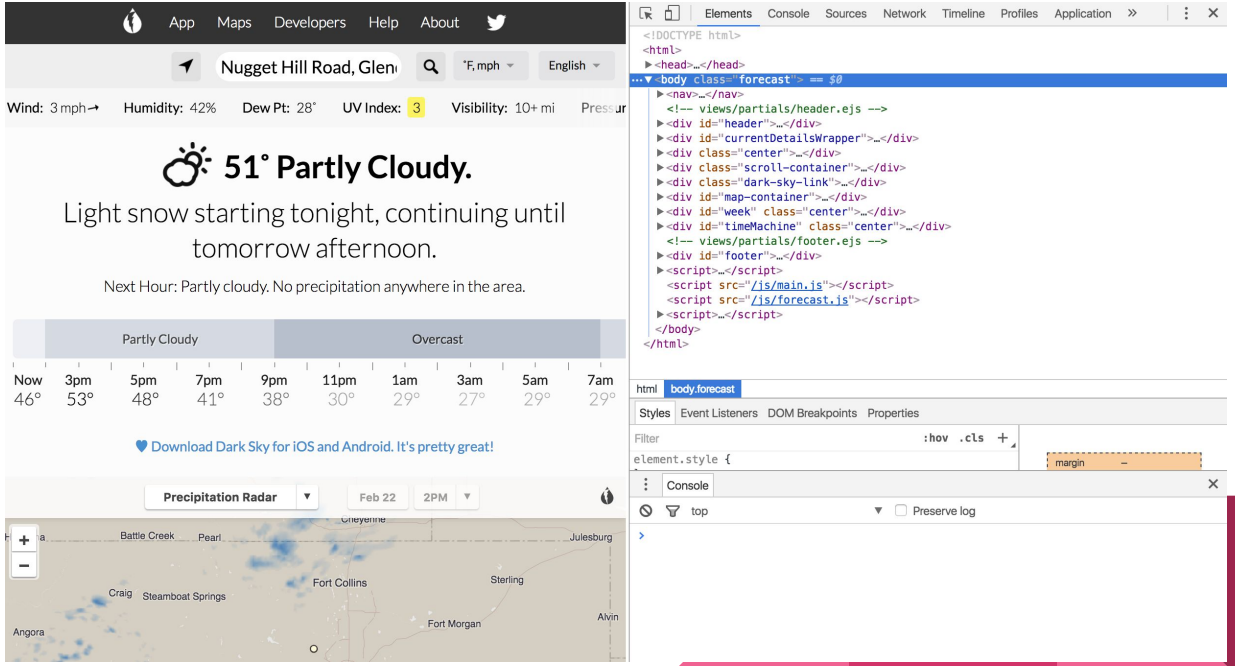


Pre-lab

4. HTML & Javascript (https://darksky.net/forecast/40.0878,-105.3735/us12/en)



Web browser -> Developer tools



Pre-lab

4. HTML & Javascript (https://darksky.net/forecast/40.0878,-105.3735/us12/en)

```
<!DOCTYPE html>
<html>
 ▶<head>…</head>
…▼<body class="forecast"> == $0
   ▶<nav>…</nav>
    <!-- views/partials/header.ejs -->
   ▶<div id="header">…</div>
   ▶<div id="currentDetailsWrapper">…</div>
   ▶<div class="center">…</div>
   ▶<div class="scroll-container">…</div>
   ▶<div class="dark-sky-link">…</div>
   ▶<div id="map-container">…</div>
   ▶<div id="week" class="center">…</div>
   ▶<div id="timeMachine" class="center">…</div>
    <!-- views/partials/footer.ejs -->
   ▶<div id="footer">…</div>
   ▶<script>…</script>
    <script src="/js/main.js"></script>
    <script src="/js/forecast.js"></script>
   ▶<script>…</script>
   </body>
</html>
```

jQuery

The jQuery library is a fast, small, and feature-rich **JavaScript library**.

```html
<!doctype html>
<html>
<head>
    <meta charset="utf-8">
    <title>Demo</title>
</head>
<body>
    <a href="http://jquery.com/">jQuery</a>
    <script src="jquery.js"></script>
    <script>

    // Your code goes here.

    </script>
</body>
</html>
```

We need to include the jQuery library in our code

(The browser will download and initialize the jQuery library before our code uses it later in the page)

We put our Javascript code here

Ajax http://demo.tutorialzine.com/2009/09/simple-ajax-website-jquery/demo.html#page1

- Traditionally webpage required reloading to update their content.
- This was hugely inefficient. Ideally, the server should only have to send the user's new messages, not the entire page.
- By 2003, all the major browsers solved this issue by adopting the **XMLHttpRequest (XHR)** object, allowing browsers to communicate with the server without requiring a page reload.
- The XMLHttpRequest object is part of a technology called **Ajax (Asynchronous JavaScript and XML)**
- Using Ajax, data could then be passed between the browser and the server, using the XMLHttpRequest API, without having to reload the web page.

jQuery’s Ajax-Related Methods

jQuery’s core **$.ajax()** method is a powerful and straightforward way of **creating Ajax requests**.

```
// Using the core $.ajax() method
$.ajax({

    // The URL for the request
    url: "post.php",

    // The data to send (will be converted to a query string)
    data: {
        id: 123
    },

    // Whether this is a POST or GET request
    type: "GET",

    // The type of data we expect back
    dataType : "json",
})
  // Code to run if the request succeeds (is done);
  // The response is passed to the function
  .done(function( json ) {
     $( "<h1>" ).text( json.title ).appendTo( "body" );
     $( "<div class=\"content\">").html( json.html ).appendTo( "body" );
  })
  // Code to run if the request fails; the raw request and
  // status codes are passed to the function
  .fail(function( xhr, status, errorThrown ) {
    alert( "Sorry, there was a problem!" );
    console.log( "Error: " + errorThrown );
    console.log( "Status: " + status );
    console.dir( xhr );
  })
  // Code to run regardless of success or failure;
  .always(function( xhr, status ) {
    alert( "The request is complete!" );
  });
```

Set the URL for the request
Send query string: id
“GET” data
Data type

The response is passed to the function, do sth...

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. JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others. These properties make JSON an ideal data-interchange language.

```
{"menu": {
  "id": "file",
  "value": "File",
  "popup": {
    "menuitem": [
      {"value": "New", "onclick": "CreateNewDoc()"},
      {"value": "Open", "onclick": "OpenDoc()"},
      {"value": "Close", "onclick": "CloseDoc()"}
    ]
  }
}}
```

For submission

- Follow the instructions from part 1 to 6, step by step
- Show TA the outputs (terminal, website)