Project 2 Encryption Demo

This is a graded assignment. Note: I will add more detail to this document by April 23rd but nothing major will change

Due Tuesday **May 9th, 2017 11:30AM** (12.5%)

Build an attractive web app that encrypts and decrypts messages using a simple Caesar Cipher.

What will be graded

HTML, CSS, JS, code style and structure, website content (clarity of text in user instructions, etc), project cover document (description, test plan, known issues).

See "Phases" section for info about Test Plan.

Constraints

Implement the project only using the JavaScript covered during Weeks 2 to 11. In other words, no jQuery.

I expect you will need to use asynchrony, Ajax, JSON, cookies, form validation techniques, key and mouse events, checking browser capabilities for cross-browser support, progressive enhancement, HTML5 semantic elements, among other things.

Functional requirements

The target audience of this app is adults (age 20-50) who use the web often but aren't necessarily tech savvy or interested in technology or computing.

The goal of the app is to teach this audience about the basic idea of **encryption** in a fun, memorable way so they can understand how their private data is protected.

It's really important to make the design simple, user friendly and easy to read.

Views

1. A page for encrypting a message.
2. A page of decrypting a message.
3. An "About" page that describes the project and, you, the developer.
4. A "wizard" sequence of instructions that shows up at the beginning if it's the first time the user visits the app. Otherwise, the instructions should be accessible via a link.

All the pages should be reachable from each other.

The "pages" doesn't necessarily have to be separate HTML documents.

Features

1. The user submits a message and a key, then is shown the result of encrypting the message with that key. Both the original and encrypted message should be shown. We only want to encrypt characters that would normally show up in an English sentence with standard

punctuation and numbers, but no unusual symbols.

2. The key represents how much to "shift" the original characters in the message. The user should be able to choose a key from a nice grid or list of available symbols/letters/characters. For example, if the user chooses "m" as their key that means that each character in their message should be shifted by 13. (You can decide how the shift amount is calculated.)
    - Encrypt the message when user hits enter, or clicks encryption key in menu. Result should update as the user types.
3. The results should be deterministic: a given message and key should always result in the same output.
4. Depending on the user's browser capabilities and settings, a different set of features should be available:
    - If JavaScript is disabled, inform them that it's required for the encryption/decryption.
    - If their browser doesn't support modern features, only allow them to encrypt/decrypt characters in the ASCII range, and ignore everything else. The keys they can choose from should only be lowercase letters and numbers.
    - In a modern browser, the keys they can choose from should all be emoji. Provide a list of at least 75 emoji. The resulting encrypted message will therefore be a sequence of emoji characters.
5. In the encryption view, the user should be able to switch to a "Encrypt According to Weather" mode. In this mode, instead of choosing a key they enter a city, then the app chooses and shows an encryption key based on the current weather in the city. For example, if the user enters Vancouver and it's currently raining in Vancouver, the encryption key should be something like the "cloud with rain" emoji character. Use a default value if the weather can't be determined. (See https://openweathermap.org/current)
    - The last-entered city should be remembered the next time the user visits the site.

About the weather

To determine the weather condition in a city, you can use the public API provided by openweathermap.org. You need to sign up for an API key. Once you do, you can make a request as below, where you replace the zeros with your API key. You will need to limit the number of requests you make and cache your data -- otherwise your API will be blocked.

```
http://api.openweathermap.org/data/2.5/weather?q=London&appid=00000000
```

The data you get looks like this. You are interested in the `weather.main` attribute.

```
{
  "coord": {
    "lon": -0.13,
    "lat": 51.51
  },
  "weather": [
    {
      "id": 801,
      "main": "Clouds", <---------- This one!
      "description": "few clouds",
      "icon": "02n"
    }
  ],
  "base": "stations",
  "main": {
    "temp": 277.34,
    "pressure": 1031,
    "humidity": 81,
    "temp_min": 273.15,
    "temp_max": 280.15
  },
  "visibility": 10000,
  "wind": {
    "speed": 1
  },
  "clouds": {
    "all": 24
  },
  "dt": 1491456000,
  "sys": {
    "type": 1,
    "id": 5091,
    "message": 0.0072,
    "country": "GB",
    "sunrise": 1491456231,
    "sunset": 1491504168
  },
  "id": 2643743,
  "name": "London",
  "cod": 200
}
```

Resources

- IMPORTANT: when testing, use their "sample" endpoint[1] or save a few results in a local json file so you don't accidentally get your API key blocked
- API howto
- API docs
- Possible weather conditions are: clear sky, few clouds, scattered clouds, broken clouds, shower rain, rain, thunderstorm, snow, mist

Phases

I suggest that you work on your project in the following order.

Phase 0

Write simple HTML and CSS skeleton for encrypt/decrypt pages. Make instruction "wizard" and about page.

Phase 1

Write a test plan: list 20 things you need to test manually in your final app to check that the features work. Try to think of edge cases and weird/bad/silly things the user might do.

Examples: Submit message with symbols, punctuation, Submit message that is too long, submit empty message, etc.

Phase 2

Simple app for older browsers: noscript fallback, no emoji, just Caesar cipher encryption

Phase 3

Decryption page.

Phase 4

Keep Phase 2 as fallback based on browser capability. Add emoji. Update output as user types.

Phase 5

Encrypt according to weather.

Grading

I'll fill this with a detailed scheme soon.

Style

Follow the JS style used in the code examples in class. The most important is to indent properly, use descriptive identifiers and **be consistent**.

All top-level, named functions must be documented in JSDoc style, as in the following examples:

```
/**
 * Blend two colors together.
 * @param {number} color1 - The first color, in hexidecimal format.
 * @param {number} color2 - The second color, in hexidecimal format.
 * @return {string} The blended color.
 */
function blend(color1, color2) {...}

/**
 * Summary of what the function does.
 * @param {string} a - The x value.
 * @param {number} b - The y value.
 */
function something(a, b) {...}
```

Points will be allocated to overall readability, code style, code structure, respect of HTML5/CSS/JS best-practices.

Submission instructions

You must submit two copies of your work:

- Upload all project files in one zip file on Moodle.
- Print a cover page and all source code and submit it **in class**. Detailed submission requirements will be described in the "Project 2" assignment on Moodle.

I will only grade and write feedback on the **printed** submission, but I will use the electronic copy for testing and inspection as needed.

---

1. For example: http://samples.openweathermap.org/data/2.5/weather?q=London,uk&appid=b1b15e88fa797225412429c1c50c122a1↩

Last update: 2017-04-06 10:36:36

Source: _javascript_423/assessment/project2_cryptemoji/spec_project2_crypto.md