Introduction

Notes to Candidate

The Nerdery Assessment Test (NAT) is a chance for you to show off your programming skills. It is a code challenge intended for you to demonstrate your development capabilities and prove your experience. While this is a relatively simple web application, we expect you, the applicant, to think of the end result as a finished project that we would eventually launch for internal use. It should be maintainable, and reasonably free from errors and defects.

What we don't expect is a high-design, high-performance driven application. You should spend upwards of 10 hours on this challenge, and we understand that this is not enough time to develop a fully robust application. You may feel free to spend additional time on it if you wish, but just know that it's not necessary to do so as long as you meet the spec's requirements. Due to the time and simplicity constraints, if you make a development decision that goes against what you would normally decide, we encourage you to include notes with your submission, and possibly be prepared to discuss your decisions in the future.

NOTE: This specification is intentionally vague in some areas because we want to see your approach to solving problems rather than dictate a solution. If you have questions about something, please ask your candidate advocate or the developer assigned to you. We are most definitely not trying to trick you or hide any information. It's possible, maybe likely, that you will see an area in which we can improve the specification and make it more clear. We'd love that feedback. We cannot, however, provide specific advice on completing the challenge.

ChangeLog

18 Sep 2014 Added Success Criteria section and ChangeLog

SnaFoo

Nerdery Snack Food Ordering System

Background

The Ordering, Catering, and Delivery (OCD) department at the Nerdery provides snacks daily in all Nerdery offices. These snacks are purchased monthly and rotated daily to give employees a wide variety of choices throughout the month. Some snacks are always purchased, but other snacks vary depending on the mood of the purchaser. Four different snacks are provided daily from a selection of ten snacks purchased monthly. OCD is open to new snack ideas from Nerdery employees.

Expected Audience

The end users of this application will be the employees of the Nerdery headquarters in Bloomington, MN. The majority of these users prefer Google Chrome, release channel, for a web browser. It can be assumed that employees always use the same computer and same browser to access the application from the Nerdery headquarters.

Project Definition

This project is to develop a prototype application for the Bloomington office that allows employees to participate in choosing the snacks to be purchased for a particular month. The application will have two pages: Voting, and Suggestions.

Voting

The Voting page will show the snacks currently being purchased or suggested for purchase for the coming month. This should consist of two separate lists: those that are always purchased and those that have been suggested so far this month by employees. Each of the suggestions should display the current number of votes for that snack this month and the last time the snack was purchased.

Each suggestion should also have a vote button that allows an employee to vote for a snack. Once an employee's vote has been entered, the employee cannot change their vote. Employees can vote for up to 3 snacks per month. The number of votes remaining for the employee should

be displayed on the page. If an employee attempts to vote more than the allowed number of times, an error message indicating that they may not vote until the next month should be prominently displayed.

NOTE: Because this is a prototype you may use cookies to track whether an employee has used their maximum number of votes. You may assume that every employee always accesses the system from the same browser/device.

Snacks which are always purchased should not display the number of votes, the last time purchased, or have a vote button.

Suggestions

The Suggestions page will display a dropdown from which an employee may select a snack to add to the list of suggested snacks for the month. The set of optional snacks returned by the web service described below should be used as the source for the items in this menu. This dropdown should not include snacks which are always purchased or snacks which have already been suggested this month. There should also be a textbox that allows an employee to suggest a snack and where it can be purchased if their preferred snack is not listed in the dropdown. Employee suggestions should be added via the web service as optional snacks for future consideration.

An employee may only choose one snack from the list or enter a new snack, not both, per month.

Once the employee chooses or enters their suggestion, they will click the Suggest button to record their suggestion. The application should ensure that duplicate suggestions are not entered. If a new suggestion is offered, the location to purchase it is required. Error messages should be prominently displayed if the employee attempts to make more than the allowed number of suggestions per month, attempts to make a duplicate suggestion, or does not enter all required data when making their suggestion.

NOTE: Because this is a prototype you may use cookies to track whether an employee has made a suggestion this month. You may assume that every employee always accesses the system from the same browser/device.

Web Service

As part of a previous project we developed a web service that allows access to the current list of snack foods to choose from, whether those snacks are always purchased or optional, how many times that snack has been purchased in the past, where the snack can be purchased, and the last date that the snack was purchased. This web service provides data in JSON format. It requires an API key to be provided with every request.

The web service allows you to get access to the current list of snacks - any that are always purchased or ones that have been previously suggested. The web service also offers the ability to create a new snack entry. The web service does not track votes for snacks but does track the last time a snack was purchased and it's location. You can assume that there is another administrative application that OCD uses to manage this data. Detailed documentation on the web service can be found at https://api-snacks.nerderylabs.com/v1/help

This web service gets taken down periodically for maintenance. In the event that the web service is down, you should provide suitable messaging to the employee so they know to come back at a time when the web service is working.

Code Expectations

Well-Documented

The application code should be created with the assumption that other developers will work on it in the future, so it should be readable and commented enough to make it easy for others to understand and maintain.

Coding Style

The application code should follow GitHub's Ruby Styleguide found at https://github.com/styleguide/ruby , though TomDoc use is not required

Error Handling and Security

All application errors should be handled gracefully and display user-friendly error messages if necessary. Submission should avoid common web security vulnerabilities and exploits.

HTML

The quality of the site design will not be judged, but the HTML should be semantic and valid

HTML 5.

Food for Thought

These questions are intended to help you think about how the application might change over time. You should NOT implement anything suggested below as it will negatively affect our ability to score your NAT. You might want to consider how it would affect the choices you make in building the application. We will probably ask you at least one of these questions if you are selected for a second interview.

Think of a feature that you'd like to see implemented in this application. How would you change the application to address this feature?

If we wanted to have separate snack lists and suggestions for each of our offices, what approach would you recommend? How would that affect how the application is architected? What changes would need to be made to the web service?

How would you improve security in the application?

What would you do differently to allow employees to use any device, yet have the same rules for voting and suggestions?

If we wanted a shopping list, which always contained ten items, what approach would you take to address this feature?