Project Documentation

## Team: Santa’s Little Helpers

### **Project Overview**

Optimal Secret Santa is a simple web application hosted on Heroku that allows a group of at least 3 people to generate gift exchange assignments. A user creates a Secret Santa group on our webpage (<http://optimal-secret-santa.herokuapp.com/>) by entering the names and emails of group members. This information is then stored in a Postgres database, our optimized graph algorithm generates Secret Santa pairings, and each group member receives an email to fill out a wishlist on another page within the application. After a user fills out their wishlist, this information is sent to that person’s assigned Secret Santa.

**Explanation of Design Choices**

Initially, we had a very ambitious design for the application including using the Google API for sign-in, incorporating user preferences into our algorithm, and using the Google Maps API to help users locate stores where they could purchase gifts for their assigned partner. However, after diving into the complexities of Python Flask, web application development, and database management, we realized that given the limited runway to complete the project, we needed to simplify our design while maintaining the core differentiators that distinguish our application from existing competitors. Moreover, we realized that forcing users to “log-in” to the site or provide information such as their location (to utilize the Google Maps API) would actually make our application less desirable to users who are concerned about providing personal information to unfamiliar online sources.

As such, we focused on a few important features:

* A user-friendly, yet not overly ostentatious interface
* An email function to automatically send information to users, including:
  + A message indicating they have been added to a Secret Santa group along with a unique link to a page to create a wishlist
  + Another message assigning the person as a Secret Santa for another group member (their “partner”) and providing the partner’s wishlist
* An anonymity-preserving random gift assignment algorithm that used built-in Python data structures such as lists, dictionaries (hash tables), and the *random* library. This algorithm worked in such a way that makes it impossible for a user to mathematically infer the identity of their Secret Santa and prevents issues such as someone assigned to themselves, and sub-cycles where A gives to B and B gives to A.
* Hosting services on Heroku to make our application universally available and an underlying Postgres database to store information needed for our application

Ultimately, we believe this differentiates our application since we not only collect minimal information, but also allow a “Secret Santa” process to be completed in an entirely virtual and anonymous fashion. Moreover, by collecting and sharing wishlist information with an individual’s designated “Secret Santa”, we provide a value-added experience to gift-givers by informing their choices while preserving their anonymity.

### **Front End**

* **Flask/HTML** - Flask is a lightweight web framework that is useful to create webpages and supports importing of additional packages to implement useful features. It also allows us to do routing of our HTML pages that are within our ‘templates’ folder.
* **jQuery** - A JavaScript library that simplifies the vanilla JavaScript language to allow more intuitive DOM manipulation. This was used to dynamically create the Group’s name and email entry and allows for more text boxes to be created.
* **Cascading Style Sheet (CSS)** - A standard stylesheet format that we used to make web pages more aesthetically pleasing.

### **Back End**

The backend was written entirely in Python. The gift exchange assignment algorithm used built-in Python data structures such as lists, dictionaries (hash tables), and the *random* library. Previously we explored the possibility of using NetworkX graph network library but found it to be computationally intensive.

**Algorithm**

We designed and implemented an anonymity-preserving random gift assignment algorithm. No person can infer the remaining gift assignments based on the fact that person *i* was assigned to give to person *j*. Algorithm and data structures research included: Graph theory, graph traversal; Vertex disjoint circuit covers; Hamiltonian Cycles; NP-completeness; Probability; Anonymity and derangements; SQL database lookup time, querying, extraction

The Secret Santa problem is described by the [Wikipedia entry](https://en.wikipedia.org/wiki/Secret_Santa):

Secret Santa is a Christmas ritual involving a group of people exchanging anonymous gifts. Participants names are placed in a hat and each person draws a name for whom they are to buy a gift. Presents are then exchanged anonymously. There is usually a gift giving occasion, where all the presents are placed on a table, with the name of the receiver, but not the giver.

The central problem we seek to solve is that using basic statistics and probability, it is possible to infer who your Secret Santa is based on the order that names are drawn from the hat. Furthermore, this process is not “fair”, such that each member has an equal probability of being drawn. In some cases, the entire process has to be redone if someone (especially the last person) draws their own name. By leveraging graph theory concepts, we can generate the optimal Secret Santa assignment. A Hamiltonian cycle is a closed loop through a graph where each node is visited exactly once. This is ideal for our problem because the directed edges from one node to another is analogous to a gift giving relationship. Additionally, it prevents issues such as someone assigned to themselves, and sub-cycles where A gives to B and B gives to A.

The algorithm works by randomly connecting nodes to other nodes while trying to form a cycle. For each element, we check against our constraints (no self assignment, no sub-cycles), and if all are valid, the list of assignments is passed onto the email sending function. The best case is that the assignment works on the first pass, while the worst case occurs when all but the last assignment is valid. On average, since we rely on random number generation, the algorithm linearly scales with the number of members.

**Runtime analysis and space complexity**

For *n* members in *k* groups (assuming each group contains exactly one member):

Gift assignment generation: Average case , Worst case

Prevent duplicate emails (compare list to set):

Verify that result is a valid Secret Santa graph:

Dictionary containing *n* items with *k* keys

List of *k* keys

Previously we implemented constraints that prevented someone of the same household/group to be matched with another but we found that imposing such constraints significantly increased complexity and the risk that the algorithm will not find a match (when one group has more members than half of all the users).

**‘Email Bot’ and Associated Functions**

We use the term ‘email bot’ to describe several of the methods associated with the email functions in our program. These include an email sending function (using flask\_mail for connection with the SMTP server and threading to send these emails in an asynchronous fashion), and a link generation function (using the URLSafeSerializer to generate a unique token and url\_for to create a URL to our wishlist page). In addition, we used the os package to conceal the password/Secret key associated with these functions as environmental variables.

**Modules/Packages Utilized**

**flask\_mail** - We imported flask\_mail extension to provide our email bot with the ability to interface with the SMTP server and send emails. From flask\_mail we imported Mail (to create a mail instance) and Message (to create a message instance) for the emails we sent to group members using our application.

**ItsDangerous**  - From the Flask ItsDangerous package (used for sensitive information), we imported the URLSafeSerializer class. This class allows us to generate unique token that we could append to links to each user in order to access the wishlist page.

**url\_for** - From flask, we imported the url\_for method to generate a link (which we would send to users) which would route them to the wishlist page (so they could submit their wishlist). We appended the token generated from the URLSafeSerializer to make each link unique.

**‘os’ package** - This package allows developers to access the hidden keys for the email client that are on the Heroku backend hosting service. By using this package, we were able to conceal our email password and URLSaferSerializer secret keys as environmental variables (stored in our Heroku configuration) rather than hardcoding them into our program itself.

**Thread** - From the Python Threading module, we imported the Thread class. This Thread class allows us to run our email sending method as a background task so that our web page refreshes instantly whenever a user submits data (rather than the page ‘hanging’ until all emails are sent as with the traditional flask mail send method or the bulk mail send method).

**PostgreSQL** - A free and open-source relational database management system that holds the data sent (name and email) from our front-end web page input. This also holds the data from the algorithm that matches a group member to their partner and the user’s gift wishlist which is sent to their partner.

**SQLAlchemy** - From the Flask-SQLAlchemy extension, we imported the SQLAlchemy toolkit to enable querying of database without using raw SQL commands in our Python script.

### **Hosting Service**

**Heroku** - A cloud-based platform-as-a-service (PaaS) that hosts various programming languages and allows users to build and deploy their website without needing to create their own web server.

### **Known Bugs**

When a group is created using a Safari web browser, the email bot will send multiple emails to users. This appears to be unique to Safari and does not occur with browsers such as Google Chrome. We tried troubleshooting this issue with Course TF’s, but ultimately were unable to resolve (we were told that Safari occasionally has issues with web applications).

The emails from the email bot will generally go to a user’s Spam folder. This may be a result of Gmail’s spam detection when sending bulk emails with similar messages.

The application is not rendering properly on mobile devices (ie the submit button is not visible on the group submission page).

Currently, if a user simply puts in the same email for all entries in the sign-up

### **Future Improvements**

In developing a second version of the application, we would implement the following additional features:

* HTML version of the emails incorporating the “snowy landscape” banner used throughout the application.
* Additional error-checking features to ensure that only valid emails are entered on the form submission page by users.
* Dynamic table creation functionality within the database to allow for user to register for multiple groups using the same email (currently, the email entry must be unique within the database).