Daniel Alpert, Dylan Farrell, Patrick Kelley

We tackled the project by using CS50 Finance as our base template and making serious alterations from that point. What we took from Finance was the framework (i.e. login, register, lougout, and MySQL) and basic CSS as well as crucial functions that we couldn't operate to our fullest extent without (i.e. query, apologize, dump, require, redirect, etc.).

The crux of our website revolves around a MySQL database. There are three tables: users, crushes, and times. The users table holds every registered user, along with their unique id, email address, hashed password, first and last name, as well as a formatted name ("firstname.lastname" so we could easily compare). The crushes table holds the user's unique id, the formatted names of all entered crushes, and a corresponding match status of value 1 if that crush has also entered the user as one of their crushes and of value 0 if there is no match. The times table also holds a unique id, along with the starting and ending time of the user's entered availability in integer military time for every day of the week.

A user registers and their identifying information is stored in the users table using a query. The flow of the design then redirects the user to the crushes page where they enter the crushes which are then stored in the crushes table. At this point, they will be redirected to the availability page. Here they enter their availability for each day through dropdown menus for start and end times of availability. We spent a lot of time working with storing formatted times (i.e. 5:00 PM) as integers (i.e. 17), as well as the corresponding hour of the week (i.e. Tuesday at 5:00 PM would be 24 + 24 + 17 = 65). While the user never encounters these differences, they are crucial to our successfully storing, modifying, displaying, and comparing date and times. We have created multiple functions for changing these formats; their names are often self-explanatory.

The heart of our program lives in the functions page. We hoped that where we actually ran our software would be simple, clean, and short, with functions that pretty explicitly state what they are doing. Many of our functions return types flow directly into the parameters of the next function we call. There are many of these functions, but a good example would be createTimeArray takes in the ID of a match and returns an array of times (array of length 168 with Booleans as values) when the match is free. findSharedFreeTimes takes in two arrays of free times (the user and the match) and returns an array of hours that both people are free. setUpDate takes in an array of hours that both people are free and returns an array of a day, time, and location that the date will happen. idToEmail takes in the id of each match and returns the corresponding email address. To pull it all together, emailMatches takes in both email addresses, the time, date, and location of the date and emails it to both users.

We actually run the matching, setting up a date, and emailing algorithm through the time.php, update_time.php, and update_crush.php pages. At this point, the user's crushes and times are stored, so we have all the information necessary to run the software. These select lines in time and update_time are the most fundamental part of veritinder.com. But as we said, much of the lines have significant calls for functions residing in functions.php.

We designated different times of the day as breakfast, lunch, and dinner, and for each of these we hardcoded in multiple restaurants in the Square that would be appropriate for that meal (i.e.

Zoe's for breakfast and John Harvard's for dinner). The algorithm randomly chooses one of the time-appropriate restaurants. So while this website is technically open to anyone, all blind dates are set up in Harvard Square. Our emailing function uses data collected from other functions to email each member of the match identical time, day, and location information for their blind date. The match's name is intentionally left out to create suspense.

A couple things we wish we could have addressed: first, while we have included a huge number of sanity checks, we have no way to differentiate between potential users with the same name. We toyed with using Harvard IDs or Facebook accounts, but that would have mandated information that we didn't have time for or access to. Another thing we weren't able to do was restrict the user from being able to update their crushes unlimited times, so that they can't change their crushes one at a time to find out who has a crush on them back. We didn't want a finite number of changes allowed; instead, we wanted some sort of timer that would allow the user to update their crushes once per week.

Finally, we thought the best place for our program to live is a public server. We want our friends, family, and instructors to be able to access it over the Internet, as a real, functioning website! We bought a domain name and transferred our code and databases onto the web.

We really hope you have as much fun using the website as we had making it.