Designing Coordinate. 

In order to create an intuitive and simple interface, we decided to make most of the website's features accessible from a single page, rather than having each of the features redirect to a separate page. We quickly realized, however, that displaying all of the features on the main page had the potential to make it cluttered, thus reducing elegance and accessibility. We therefore opted to use JavaScript to dynamically change the HTML of our home page based on the user's actions. Whenever a user clicks a button, further information appears on the same page, rather than redirecting to a new page. The functionality of the bulk of the site could therefore be centralized in a few files, and users would be able to access all features on one centralized page.

The implementation of Coordinate involved storing a lot of data, which we were able to do server-side in a SQL table. We created a SQL table that kept track of user accounts, enabling them to log in and interact with one another. The friendships between users were kept in a centralized "friends" table, which could be queried to determine a specific user's friends. To enable users to send their coordinates to each other even if they were not simultaneously logged in, we also stored user's coordinates in a SQL table, so we could access and update them at any time. In order to conserve space, each table contained only the minimum information necessary. For example, in the SQL friends table, we only stored the IDs of each user and his friends: the names and emails of the friends could be found later by querying the users table. The request table, on the other hand, required storing many parameters like the date, message, and sender/receiver of each Coordinate as well as the target user's location. In order to determine the formatting of the request in the "Coordinates" tab on the website, we also had to store whether each Coordinate was a "share" or "request" and whether it was read or unread. In an effort to save space, entries are deleted from the requests table if the request was ignored.

Since Javascript is a client-side language that cannot easily interact with server-side SQL tables, we used PHP in order to query the SQL tables. However, variables in PHP cannot be directly transferred to Javascript, so we utilized separate methods for each direction of transport. In order to get a variable from PHP into Javascript, we created our own APIs using the json_encode function. This was used to make the friends and requests SQL table, which kept track of contacts and Coordinates for a user, readable in Javascript. We also needed to transfer variables from Javascript to PHP. We did this by creating HTML forms that took on the values of the variables. When the form was handled by a controller, the controller had access to those variables under the POST variable. For example, geolocation is best called in Javascript, but we needed to insert the position into the "requests" SQL table. Therefore, whenever a user allowed access to his coordinates by clicking the accept button, we sent his latitude and longitude in a hidden form. Whenever a user needed to interact with the SQL tables, we created an HTML form that was handled by a PHP controller that could query the SQL tables.

The end result is a smooth and simple interface that allows users to interact with the website without redirecting to other controllers. Although much of the back-end work of handling stored data was done in PHP, we used Javascript in order to efficiently create a centralized front-end experience that could communicate with Google Maps API and retrieve geolocation data.