**Summary of SQL Refactoring by Dimitri and Jeremy**

1. Got rid of most functional code found in most pages since it was very difficult to follow with no comments on logic.
2. Refactored logic from functional code into cohesive classes that deal with certain aspects of the business logic; e.g. checking for conflicts on a new reservation
3. Refactored TDG and Mapper classes so that they conform to specific interfaces that follow basic CRUD functions. We used inheritance and abstract classes to increase reusability between mappers and TDG objects. Mappers now build Domain Objects based on requested queries from the TDGs and the data returned by the TDGs.
4. Replaced all SQL queries. These queries were vulnerable to SQL injections as their parameters were not sanitized. Incorporated the DBAL framework that wraps SQL queries into objects. The DBAL framework wraps PHP’s native PDO library which insures clean and safe queries.
5. Added a settings file that can be parsed by a singleton object in order to quickly retrieve app-wide settings. This made it easier to reuse settings.
6. Removed redundant php sessions that might have caused session clashes would could potentially log the user off for no reason.
7. Cleaned up javascript by using the already built-in jQuery library. Most javascript was converted to jQuery in order facilitate Ajax calls between the UI and the backend. This allows for future modularization of jQuery code.
8. Added PHP autoloaders that allow classes to be instantiated and found at runtime rather than calling file imports across the application. This allowed for any object to communicated with other objects with having to include the class’ file destination.
9. Cleaned up much of the UI using jQuery modals rather than the included Bootsrap modals. This allowed to us to use jQuery functionality mixed with the modal windows as they were all based on the same jQuery library.