Food Waste Prevention - Requirements

11/04/19

**Group 3**

Haelyn Brandt

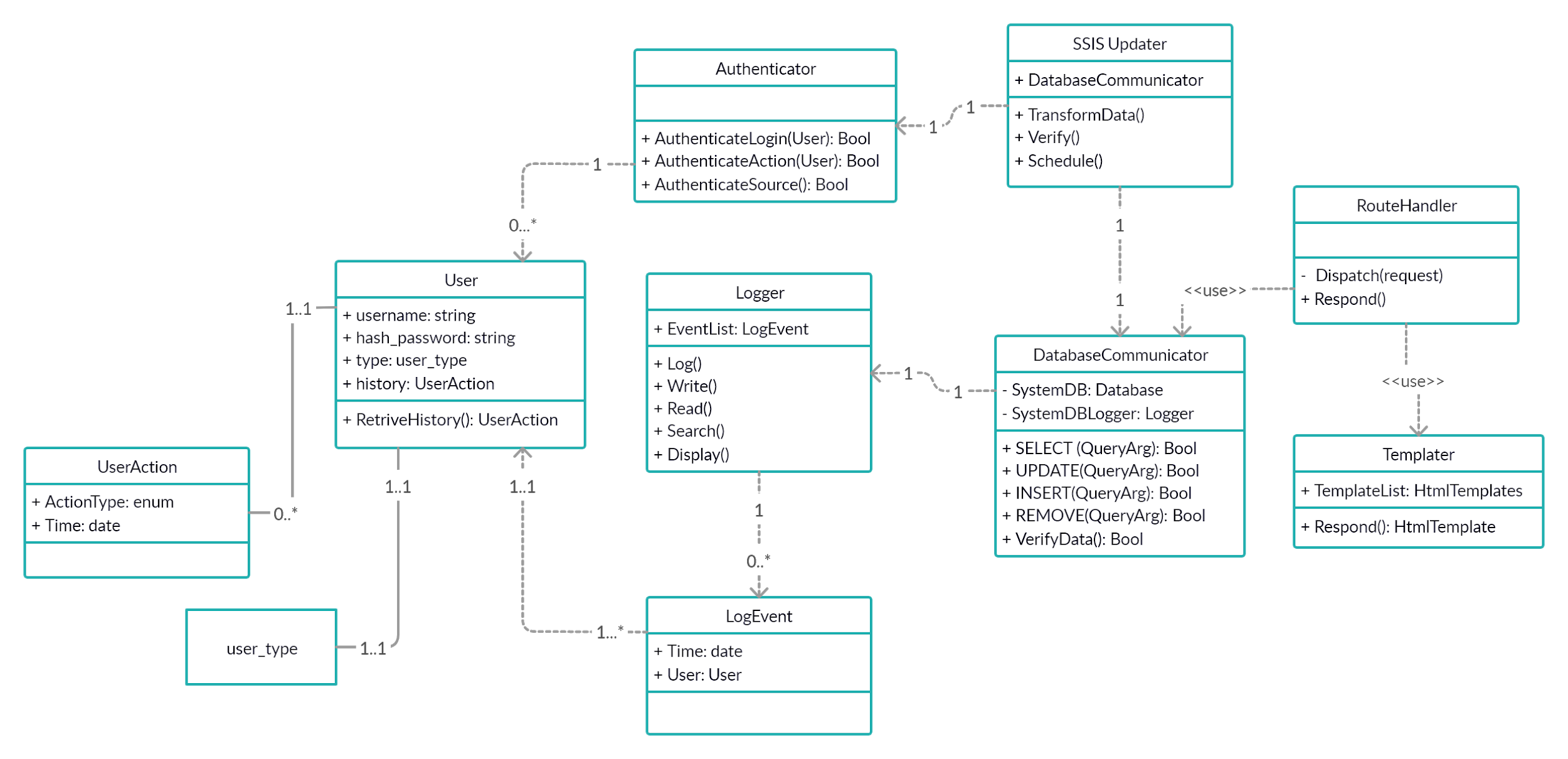
Martin Edmunds

Neil Gayeta

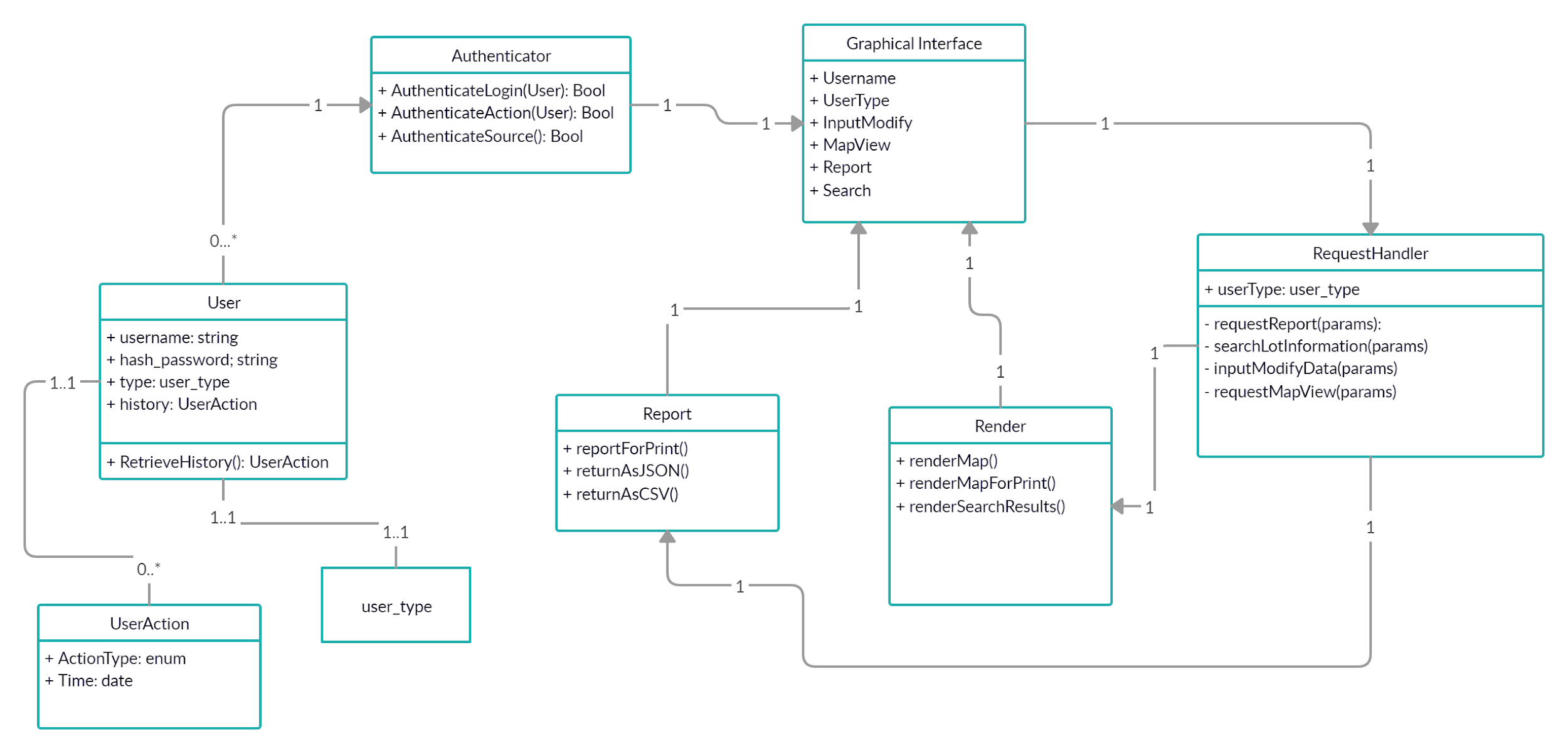
Brandon Goza

**UML Class Diagram**

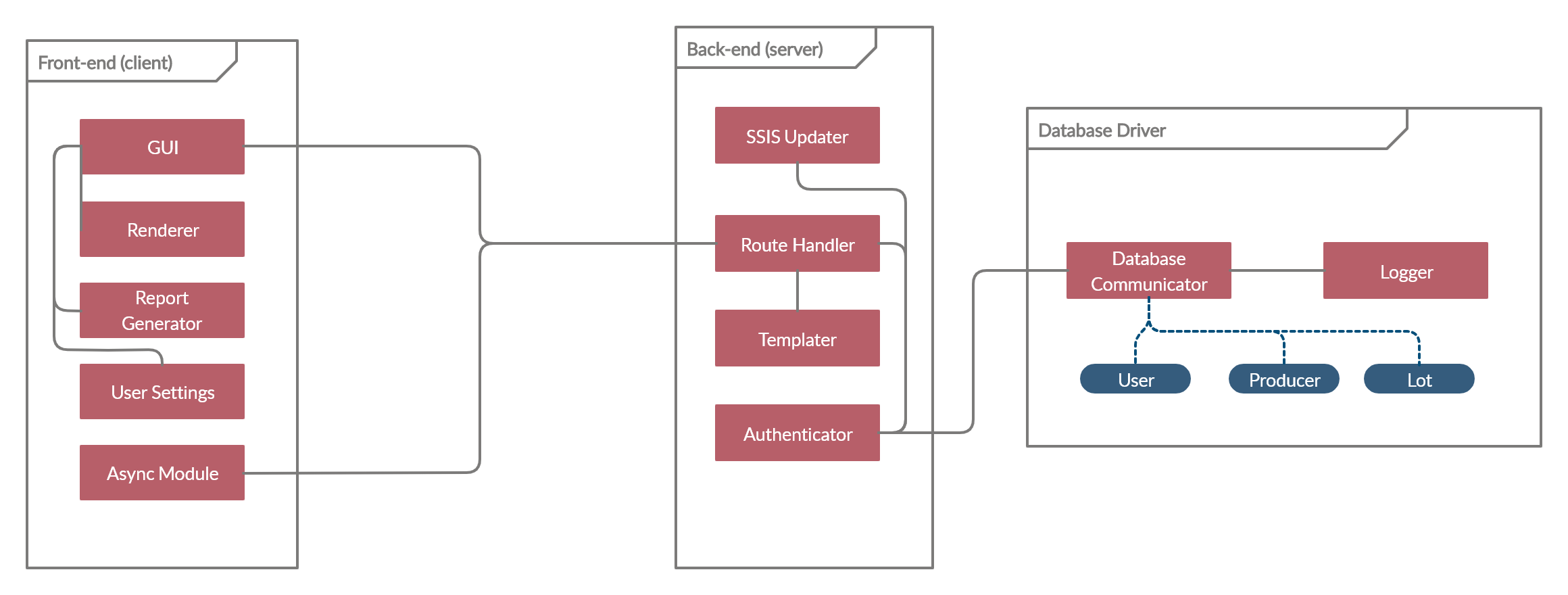
**Server**

****

**Client**



**System Package Diagram**

****

**Packaging**

* **Coupling**
  + **Data Coupling**
    - **Our system, both the client and the server contains several instances of data coupling. On the client side, almost all data received from the server will be passed directly into the following modules: the map renderer, the report generator, and the GUI. Although this data will be structured via JSON, most of the interpretation will be done in the specific modules themselves.**
  + **Stamp Coupling**
    - **Regarding the server’s connection to the client, the server will be delivering a structured HTML payload via the templater. Along with HTML, the client will have to be served the necessary code to implement the renderer and map generator.**
  + **Control Coupling**
    - **For control coupling, our goal was to minimize the cross-module function calls; however, when designing a GUI, avoiding control coupling is almost impossible. The GUI module will be involved with calling the other modules to do specific tasks, such as render the map or change user settings.**
  + **Other Coupling**
    - **Our design excludes both content coupling and common coupling from the client. In regards to the server, we separated the server-client connection from the server-database connection; this avoided situations in which the SSIS Updater module, and the Database Communicator were reading/writing the same data.**
* **Cohesion**
  + **Function Cohesion**
    - **Our design for the client, server, and database components prioritize maximizing these form of cohesion. All modules that request access to database information first have to work with the database communicator module. For the client, the async module must work with the route handler and templater to serve a functional web page to the client.**
  + **Communicational Cohesion**
    - **Several modules on the server share the same data. Output from the SSIS Updater should match our database schema. The authenticator should package both user requests and SSIS requests such that database lookups can approve or deny requests based on a user’s account type.**
  + **Procedural Cohesion**
    - **Regarding procedural cohesion, all of the back-end modules work together with other client or database modules to accomplish tasks. However, some modules such as the logger and authenticator rely on procedural execution. The authenticator and logger modules are activated when one module requests information from another module.**
  + **Temporal, logical, and coincidental cohesion**
    - **Our design attempts to eliminate these undesired forms of cohesion. The modules in the client and server are packaged in such a way that if any two modules communicate, they do so for a related purpose. In our previous data flow diagrams, it can be shown that our modules should never be in an ambiguous execution state.**

**Design Assessment**

Based on the requirements and the potential for code reuse, both incremental development and iterative development would equally support the design of this system. The system analyzes data to provide reports but these reports cannot be produced without shipment and food distribution data provided by the users.

The main function of the system is to analyze the data and provide the results to the main map in graphs or charts and also as printable reports. The incremental development would be best suitable because in order to provide accurate reports, the system needs to analyze data before being able to provide data.

The iterative development is just as suitable because in order for the system to perform analyzation, the system requires customers or users to automatically import or manually upload data. New data would need to be uploaded/updated every time there is a shipment by a producer or distributors and as the database grows, usability will continue to improve based on importance of certain data compared to others or what the customers are looking for.

**Useful Design Patterns**

**Adapter:**

Since data is uploaded or imported from different food producers/distributors, adapter pattern would aid in synchronizing the functionality between newly imported data and existing data on the system. The data is continuously imported in by various users so we need to make sure that the system is able to interface with these data.

**Facade:**

The main function of the system which is producing the main map along with many other functions in the program heavily utilizes the graphical user interface which acts as a unified, high-level interface to the various subsystems such as analyzation of food data and producing graphs/charts and reports. As we can see from previous assignments, the user will have access to things like print report, view map, view charts and other data uploading and editing functions but are kept from accessing the complex functions such as pulling back shipped distributions, assigning unused food to other areas or deciding new distribution amounts in each region. The user will be able to import their data and see the reports of how all data were analyzed to self adjust their distribution amounts but will not interact with underlying complex functions of the system.

**Interpreter:**

Similar to reasons for having adaptor design pattern, interpreter pattern would be useful because the system will require users to continuously import/upload/update data in order to analyze and produce reports/data. The data automatically imported or manually uploaded by users may all be in different form and will might need to be “interpreted”.

**Observer:**

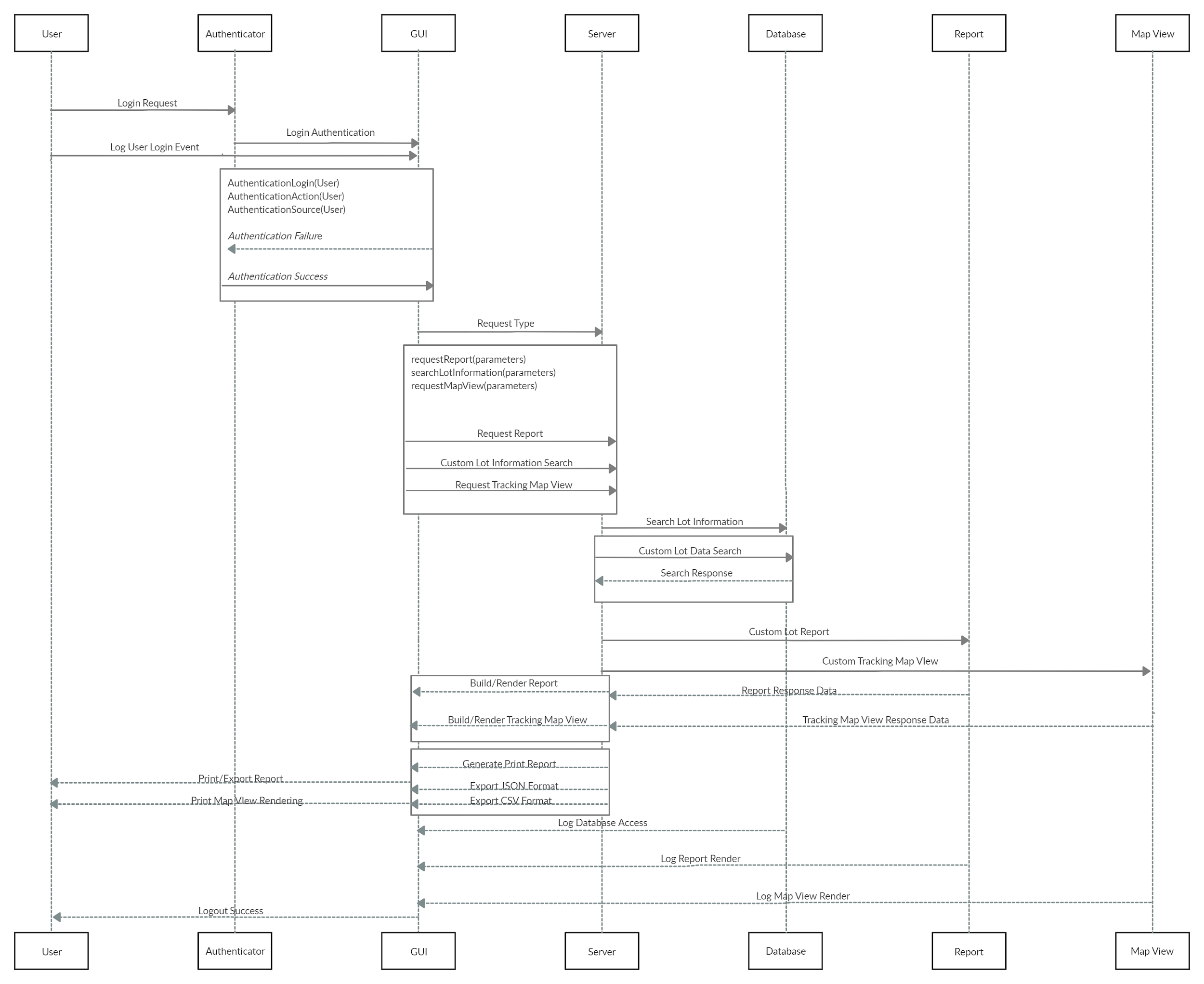
One of the main functions of the map is to color shade the regions based on food distribution. The map will need to track real-time data and update UI according to changes based on data analyzation. As stated in lecture, it “watches for another object to change state, maintains a list of observers and notifies them”. This will change graphs and color shaded regions on the main map and provide the users with the most up to date information about food distribution and waste status.

**Decorator:**

This design pattern would be useful because in lecture, it is stated that decorator can be “thought of as a role-based system so that you define roles and then those instances are appropriate to that role”. The producers/farmers will have different functions available than other users such as regional distributors or third-party organizations.

**Use Case Sequence Diagram**

**Use Case Chosen for Sequence Diagram - *Use Case 3*:**



**Interfaces**

**Computer location services:** Input: call to get current location data. Output: current location coordinates to allow for mapping.

* **Inter-package interfaces:**

**GUI:** Input: data from the other packages’ output will be sent to the GUI for display to the user. Output: the GUI

**Authenticator:** Input: user login id, password, type of account. Output: a REST request to the authentication server containing user login id, password, and account type

**Report generator:** Input: user-specified criteria (date range, type of food, region, etc.). Output: formatted queries to be sent to the Database Communicator, result set returned from database communicator interpreted and forwarded to the GUI in the form of maps/charts/raw data.

**Database communicator:** Input: front-end generated queries to be run against the database, lot data uploaded from producers/distributors (via SSIS). Output: the result set of the run queries to the report generator and the logger, status of the data uploaded from producers/distributors to the GUI and the logger

**SSIS:** input: Lot information data from local database in native format to be transformed into readable data for the database communicator. Output: transformed data in the pre-prescribed format for uploading lot information to the database communicator

**Exceptions**

**External Exceptions:**

Some examples of these could be user having trouble logging into the system or uploading data to the system due to connection errors. The user would be asked to repeat the login process and if continue to fail, ask to check their connections. Same procedures apply for failure to upload data and user can also be asked to check their local system or the data file. If all options exhausted, the support team would review and see if further assistance could be provided.

**Internal Exceptions:**

Handle all cases but make sure edge cases that may be rare are handled programmatically. When these edge cases occur, exception handling is implemented to ensure the data flow of the system does not halt and continues execution and responds with an appropriate response that is handled by the client gracefully and no chance of unhandled behaviors occurring and disrupting the expected flow of data. An example of one of these errors that needs to be handled can be when a report request, search request, or a tracking map view request is made and an external error occurs where the server loses connection to the database and an error is returned back to the graphical interface. The interface may not have anticipated the response and if the response was not handled an exception will be thrown and an unexpected response will be displayed. We need to ensure at each level from client request to server interfacing with the database, is handled for any potential exceptions being thrown when an unexpected loss of connection occurs (i.e. database connection loss).

**Team Member Contributions**

The following bulleted-list summarizes team member contributions for HW4 deadline:

* All - continuous team meeting and communication via Slack throughout the week.
* Haelyn Brandt - design assessment, useful design patterns, exceptions
* Martin Edmunds - UML class diagram, packaging diagram, coupling and cohesion discussion
* Neil Gayeta - UML class diagram, use case sequence diagram, exceptions
* Brandon Goza - had the flu this week and was unable to contribute more than the Interfaces section, will compensate next week