Food Waste Prevention - Requirements

10/30/19

**Group 3**

Haelyn Brandt

Martin Edmunds

Neil Gayeta

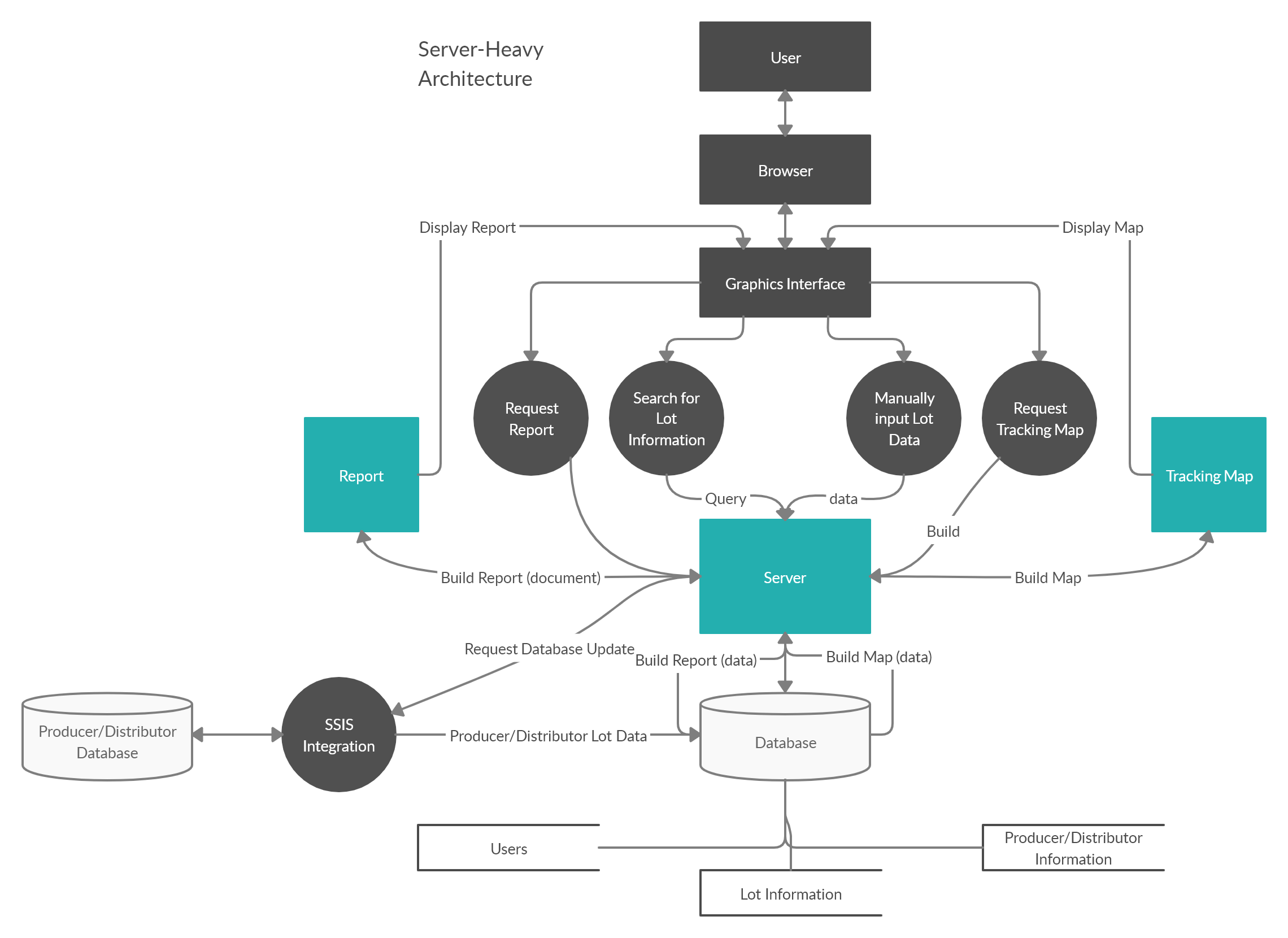
Brandon Goza

**System Architectures**

**Architecture 1**

Dataflow Diagram 1

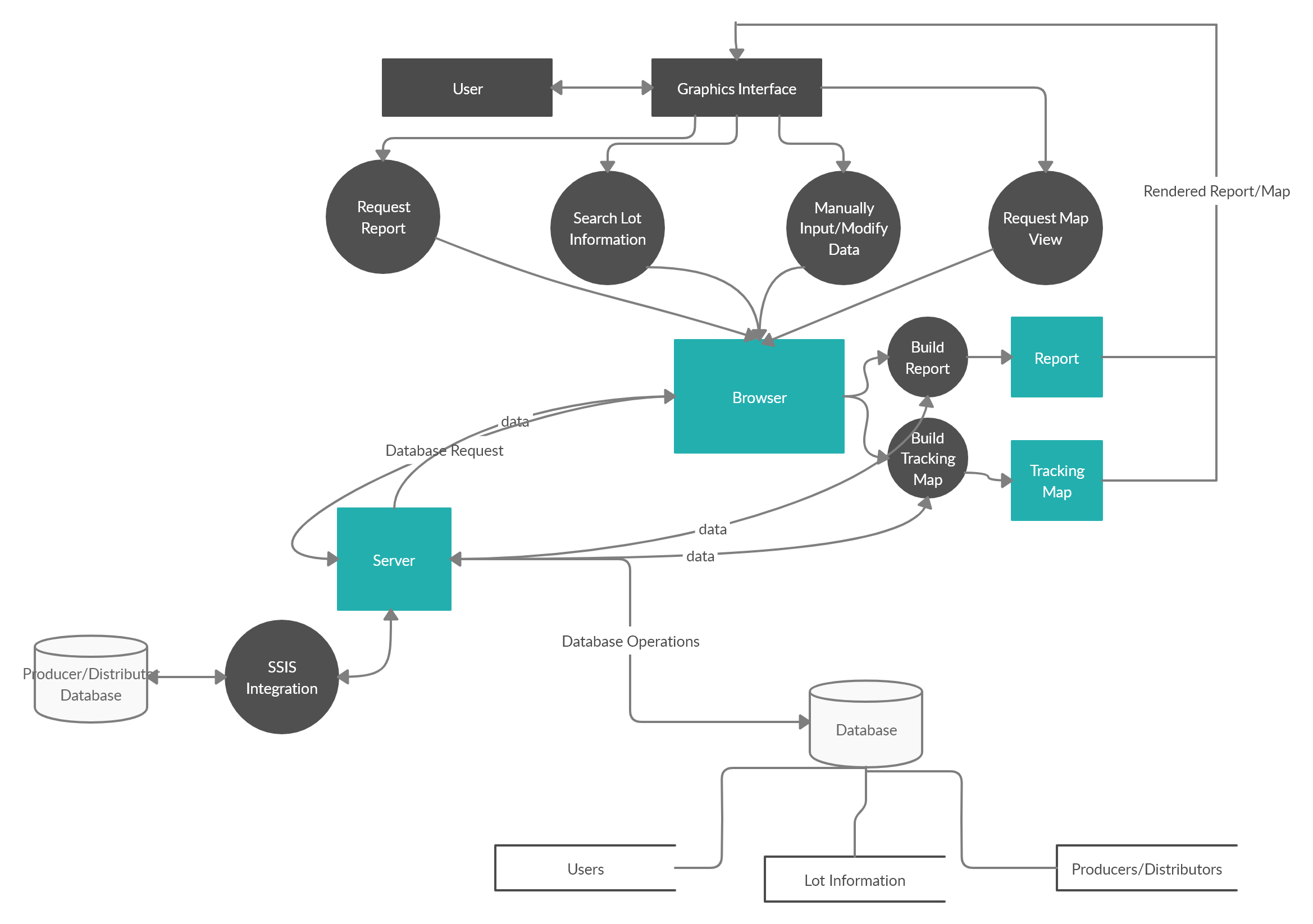
*Server Heavy Architecture*



**Architecture 2**

Dataflow Diagram 2

*Client Heavy Architecture*



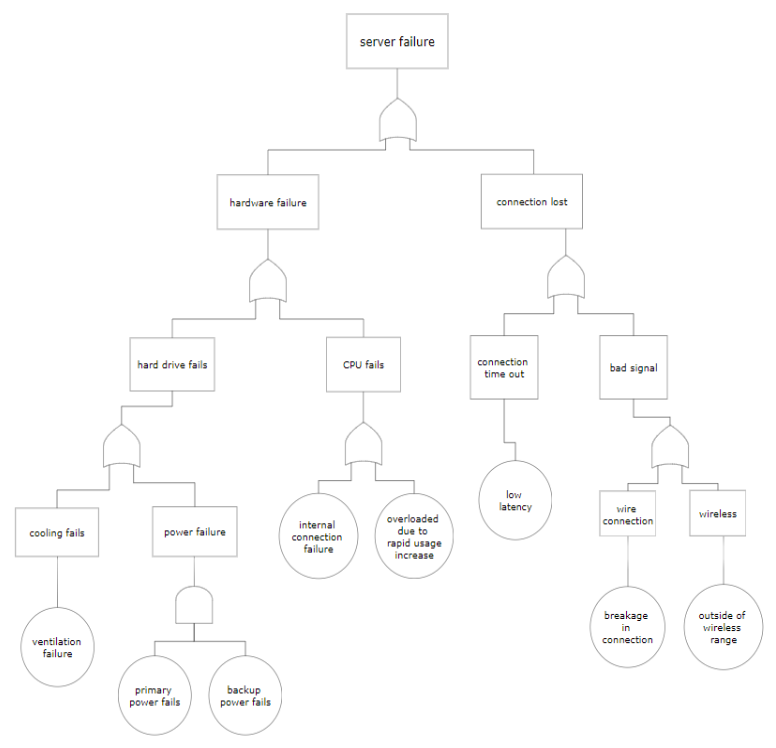
**Key Quality Attributes**

* **Reliability**
  + *Architecture #1 (Server-Heavy Architecture)*
    - A server-heavy architecture will maintain constant communication between the food loss client and the server. This connection is required as the server will be querying the database and relaying the contents to the client. However, being required to maintain a consistent connection between client and server means if the connection is loss no data will be processed, and the server will be unable to take requests for querying the database and relaying that information back to the client
  + *Architecture #2 (Client-Heavy Architecture)*
    - For a client-heavy architecture, a constant connection is not required, and up-to-date food loss data is retrievable with the client relegated to perform more processing for database requests and building of the tracking map and reports. This is possible with a slow connection. The client will utilize more local resources as the server will not be tasked to perform tracking map and/or report building (the client will be handling this in our client-heavy architecture). The server will only handle database operations.
* **Efficiency**
  + *Architecture #1 (Server-Heavy Architecture)*
    - A server-heavy architecture in terms of efficiency would be relatively less efficient compared to a client-architecture. The client will not be handling the building of reports and tracking maps which means less load is inflicted onto the client’s resources, resulting in an increase in accessibility for all users. In turn, more processing occurs on the server-side. The server is handling search requests, report requests, taking in input data for persistence, tracking map requests, and database queries. Any interruption in the (constant) connection between client and server, will have a more pronounced effect on the efficiency of the system.
  + *Architecture #2 (Client-Heavy Architecture)*
    - A client-heavy architecture would perform better in terms of efficiency in processing food-loss data, from manual input to the requesting of reports and map views. Food-loss data may be archived and stored on the client’s local storage which allows efficient retrieval and rendering of report, map, input requests. Less load will be inflicted on the server, allowing the server to focus solely on database requests and interfacing with SSIS to access the Producer’s/Distributor’s database. The constraint with this is a client-heavy architecture is resource-heavy on the local level. Efficiency will be dependent on the amount of resources available at this level.
* **Portability**
  + *Architecture #1 (Server-Heavy Architecture)*
    - A server-heavy architecture has the advantage in terms of portability for the food waste system. Because food-loss data processing and the system functions from building reports, rendering views, querying database, and interfacing with SSIS for producer/distributor database access occurs primarily on the server. Less resourceful machines have access to utilizing the system without worry of overloading their own local resources.
  + *Architecture #2 (Client-Heavy Architecture)*
    - A client-heavy architecture is at a relative disadvantage compared to a server-heavy architecture for portability. Due to more processing required on the client-side, fewer workstations may be able to utilize the food waste system due to software and hardware constraints on resources. The system may still be used, but these constraints also impact reliability and efficiency due to bottlenecks that may occur server-side or client-side. Hardware and/or software bottlenecks may occur when submitting search requests for lot information (client-side) to getting the response from the server for a custom food-loss report (server-side).
* **Flexibility**
  + *Architecture #1 (Server-Heavy Architecture)*
    - Due to the majority of the food waste system’s functionalities and data processing occurring on the server in a server-heavy architecture, modifications to the system will not require clients to perform any of their own modifications (i.e. GUI). With constant communication required between the server and client, if there is a change that needs to be deployed there may be potential for downtime. If there is downtime all clients will be affected during this time as constant communication is required to submit search requests and build reports/tracking maps. Any data processing will be interrupted until the server is back up.

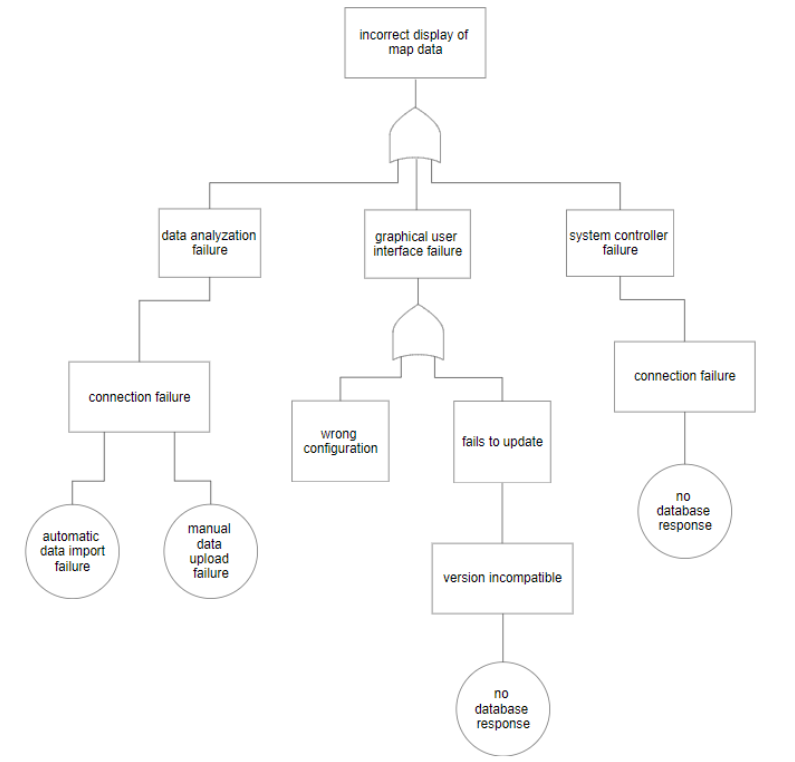
* + *Architecture #2 (Client-Heavy Architecture)*
    - A client-heavy architecture has the flexibility to perform many of the food waste system functionalities without a connection to the server. Food-loss data can be cached and stored from previous connections to the server locally allowing for quick custom food-loss report building and rendering of filtered region/lot maps. On the other hand, the ability to support different platforms and ensure integration is seamless between the food waste system and with every one of those platforms becomes a more cumbersome task.

**Failure Modes Fault Tree**

**Server Failure**



**Incorrect display of data on main map**



**Server Failure**

The architecture that is probably more prone to failure for server failure fault tree is the server-heavy architecture. As we can see above in the server-heavy architecture dataflow diagram, a functioning server is needed to access the database to build necessary parts such as building the map and making reports. If the server were to fail,

the software would be unable to function. However, in the client-heavy architecture, reports, data, and map views are produced from the browser which guards it from problems caused by server failures.

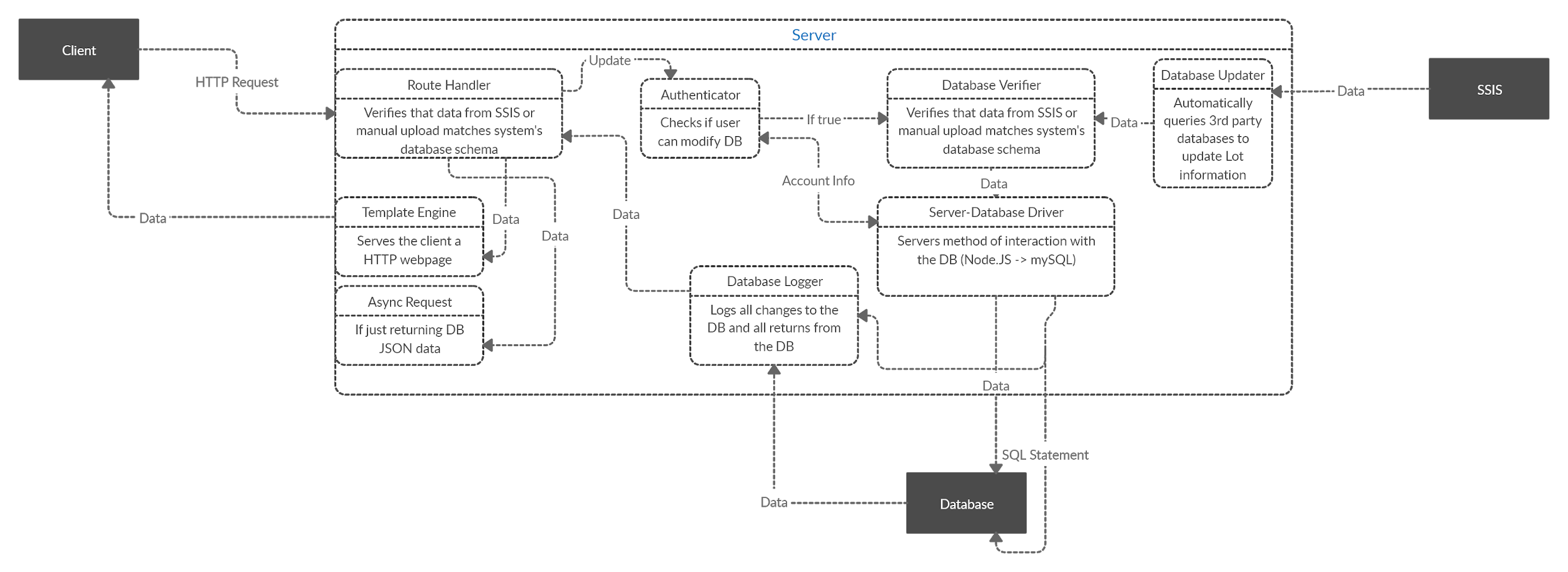
**Incorrect display of data on main map**

The architecture that is probably more prone to failure for the incorrect display of data on main map fault tree is also the server-heavy architecture. As we can see above in the server-heavy architecture dataflow diagram, the graphics interface relies on server to search and request data from the server. Server failure would cause both data analyzation failure and graphical user interface failure. In the case of data analyzation failure, we are unable to request database update, therefore unable to correctly analyze data. In the case of graphical user interface failure, we are unable to manually or automatically input lot data into the server. This would cause malfunction in requesting report form, searching for lot information, or obtaining the most updated data and ultimately causing incorrect display of data.

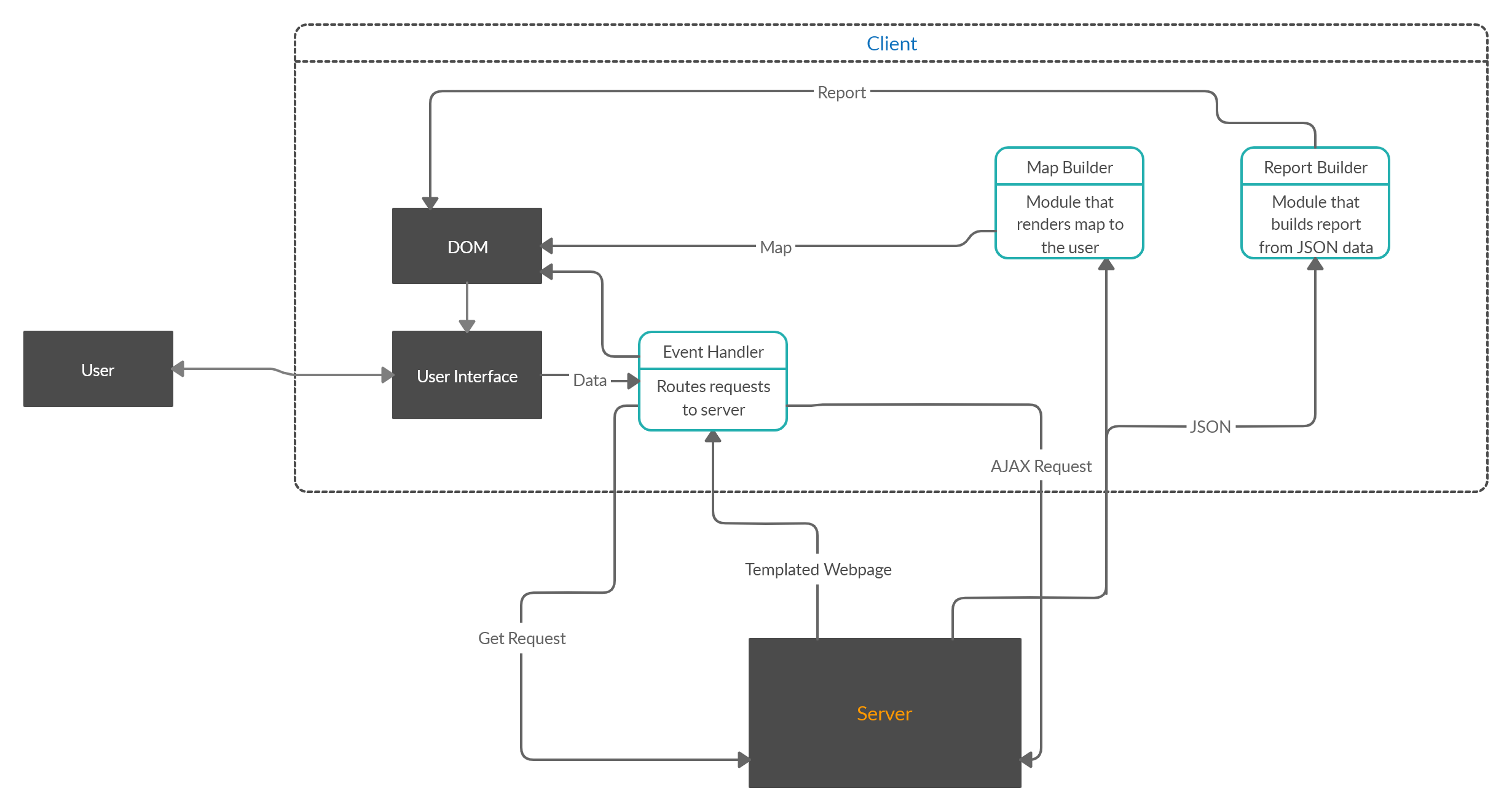
**Architecture Decomposition**

**Important Element 1**

Server Controller Lower-Level Dataflow Diagram (Client-Heavy)

****

**Important Element 2**

Client Controller Lower-Level Dataflow Diagram (Client-Heavy)****

**User Case Walkthroughs**

**User Case 1: Manual Lot Input**

* The user will begin by entering their password into the start screen. Even though we are using a client-heavy system the user will first be authenticated by the server backend. The server will match the user/password combination to the user data entity in the database. This first step would be mirrored in the server-heavy system.
* The server will then 'serve' the client the necessary home page depending on the account type. For use case 1, the producer/distributor home page will be served.
* Upon receiving the home page, the user will be presented with a graphics interface via the client-browser where the user can select an option for manual lot input. In this instance, the user will be able to select if they are modifying or uploading new data and for which producer the data is connected to.
* At this step, the client will ask the server to verify that the user account has an existing connection to the producer/distributor they are modifying. This step will ensure that the account has the necessary privilege to modify the data corresponding to the producer/distributor.
* In the case of adding lot information, the data will be forwarded to the server. The server will then feed the data into the database verification module to ensure proper parsing of the data. This parsing will be buffered and performed on the server to ensure that the upload of data is complete and hasn't been interrupted.
* Upon completion, the server will indicate to the client success or failure.
* In the case of modifying lot information, the client will request all information about a certain lot. In the client-heavy system the client will receive only the returned JSON data and proceed to build a webpage based on the received data. In the server-heavy system the JSON data would populate in a template engine and the entire html page would be served to the client. Because we're opting for the client-heavy system, the server load will be significantly reduced by having to only return lot information.
* In either case, upon a successful addition or modification, the server will query the database via mySQL queries to update the lot information. The server will then send the request and result of the database change to the database logger module.
* The user should be able to research for the modified lot information

and see the changes reflected in the database.

**Use Case 2: Normal User - Monitor/Evaluate Food Waste**

* With either architure, the user will be prompted for credentials, which will be authenticated directly against the server database
* The user will be presented with the appropriate home page, initially displaying the map view for the normal user, which will allow for both viewing food waste trends and defining new regions for the chart/report/data import views.
* Under the chart view, the user will have the option to filter the data by various criteria, including region, food type and quantity, date range, etc. Graphs will then be rendered based on these parameters, based on data actively queried from the central database. Sets of parameters may be saved on the local machine for later viewing/manipulation.
* Under the report view, the user will be given the same search options as above, returning a selection of graphs and charts. Reports are customizable and formats saved to local system. They are rendered locally based on data actively queried from the database.
* Under the raw data view, the user will again be presented with the same search options and will be given the option to save the results from the generated queries as JSON or CSV files for viewing/analysis/manipulation by outside software.
* In general, this use case presents the heaviest load and dependency on the server, as its main function is to retrieve customizable data from the database on the fly.
* Queries must be stripped of sensitive data such as individual producer name, etc. These queries must be validated/created server-side to ensure their integrity.

**User Case 3: Utilize Food Waste**

* User utilizes a device which interacts with a browser-based graphical interface and performs user authentication of a third-party organization account type at the server level (Our chosen architecture is client-heavy, but this component is handled by the server). The authentication process will ensure the user provides a combination that matches an entry in the database. Once authentication is successful, the server will serve up to the client the appropriate home page for the third-party organization account type which is the account type for use case 3.
* Once the server-side serves up the home page for third-party organizations which includes a graphical interface to interact with, the system will then present on the home page the ability to put in report requests, custom search and filter of lot information via different fields, and map view requests
* The architecture has the system perform processing of report and map view requests and the building of report and map views client-side (GUI) along with the filtering of these requests and rendering of report/map.
* The system has the server-side take in the report requests and interfaces with the database to produce data for client-side to prepare report for download (JSON, CSV). The report views provides the option to manipulate/filter which will be served up by server after querying the database and the client-side will building render the updated report.
* The system being client-heavy allows for lot information data to be archived and stored allowing for third-party organizations to have the graphical interface build and render reports and map views from local data. When the client connects to the server and requests are made causing the data to sync, the server may update the custom reports and/or map views locally for more up-to-date reports and map views.
* For the report view, the system has the server serve up contact information (email, phone number) that the GUI will include in the build and render of the report. It is included in the map view as well allowing third-party organizations to potentially capitalize on edible food loss and providing their services (i.e. pick-up and distribution).

**Implications**

When deciding on a system architecture, we had several main objectives some of which include:

* -Delivering a web-based solution
* -Control access to backend database
* -Reduce server strain

These objectives heavily influenced our decision to implement a client-server based system.In this architecture, we are able to develop a controlled system in which security, access, and traceability can be established. By moving map rendering and report generation to the client, most operations performed by the server are database lookups and transmission of JSON data. This should ensure that the server is under a light load relative to each client. One important part of our system is gathering and storing data from several producers/distributors in a single place. Since this information could possibly be restricted or be sensitive, our system needs to ensure that this information is kept secure. The client-heavy architecture ensures security by having verification implemented on the server level; all requests for information can be monitored and logged server side. A disadvantage of the client-heavy architecture is since map and report processing is performed client side, unencrypted data could be accessed when sent to the client. Ways to prevent this include limiting the data that is sent to the client upon map generation, perhaps by removing producer information from the lot data. When generating the report, since confidential information will be sent from the server to the client for client side processing, this data should remain encrypted during transmission.

**Weekly Meeting Summary**

This week, all the team members were extremely busy so we had to do most of our communication independently on slack and add in our parts of the work into the document for the other members to review. Working together was tougher this week and put our work behind for the week due to lack of communication. All the group members agreed that for future homework, we will be declaring the parts that we will work on early in the week and start discussions early in case that all group members might not be able to meet on slack at the same for project discussion to prevent ourselves from falling behind.

**Team Member Contributions**

The following bulleted-list summarizes team member contributions for completely homework 3:

* All - continuous team meeting and communication via Slack throughout the week.
* Haelyn Brandt - Failure mode fault tree and summaries
* Martin Edmunds - Use Case 1 walkthrough, system architecture 1 and 2 dataflow diagrams, architecture decomposition: important elements 1 and 2, implications
* Neil Gayeta - Key Quality Attributes, Use Case 3 Walkthrough
* Brandon Goza - Use Case 2 Walkthrough