##### **Token Management**

Tokens are generated at the start of the simulation. Virtual devices are deployed in the browser. They a

re required to provide a valid token before they can be allowed to be registered in the simulation environment (join and communicate within networks). The token entered is then be marked as “used” within the database. Any time someone tries to use a token, that the the token is unused is checked after the token is deemed valid.

Event flow for sequence diagram:

* The token is sent to the server
* The simulation manager routes the token to the token manager for authentication
* The token manager then checks with the database to ensure that the token is a valid token
* Token is checked to make sure it hasn’t been used
* The validity of the token is then returned to the client side to handle

**Token Distribution:** We hypothesize three ways to distribute the tokens. The administrator (the person who creates the simulation) would select the type of token distribution that they would like to use at the start of the simulation. For iteration 1 we chose to only implement email token propagation as we believe this to be the most sought after method of token propagation. We believe the other token propagations are not risky, as they do not satisfy any of the three Q’s of architecture, therefore we will not implement these until a later iteration.

* Email: Tokens may be propagated to devices via an email link. These emails will be generated by a “TokenPropagator” object and sent to a list of email addresses supplied to the simulation. These emails include a unique link to the server, encoded by the token, which when accessed for the first time registers the token to the device. If the link is pressed after the first time, an error message will be displayed saying that the token is already registered.
* SMS text message: Tokens may be propagated to tdevices via a link in an SMS text message. These text messages will be generated by a “TokenPropagator” object and sent to a list of addresses (phone numbers) supplied to the simulation. These text messages include a unique link to the server, encoded by the token, which when accessed for the first time registers the token to the device. If the link is pressed after the first time, an error message will be displayed saying that the token is already registered.
* Token distribution upon accessing the simulation website: Tokens may be propagated to devices which access the simulation website. This will work as follows: A device accessing the website of the simulation will be given a unique unused token to the simulation if the number of tokens to be distributed has not been exceeded. The token will then be registered to that device and will allow the device to access the simulation on all future visits.

**Token Saving:** A token, once registered to a device, will be stored as a unique key in the local storage of the device and that browser to which it is registered. A copy of that unique key is also stored on the server side in our database. This allows the keys to be compared in order to verify that this is in fact a valid key. As the key encodes the device and browser it is registered to, only that device on that browser may use it. It is browser specific as the key is stored in the local storage of that browser on that device.

### Device and Network Iterator

### In order to handle the replicated data type we required an iterator to iterate through the devices and networks. In order to have our code as reusable as possible, we created a ubiquitous “iterator” class, which upon creation takes a collection, which would be either the list of devices or list of networks provided by their respective manager classes and allows iteration through these collection. Using only a single iterator class mirrors an interface in object oriented programming languages and allows us to have more refactorable and cohesive code, providing structure to our project.

### 

### Test Scripts and HTML5 Applications

The following is our proposed approach to the distribution and application of test scripts and HTML5 applications. As this is not required for this iteration, not part of the boilerplate code for our server, and not as risky as the other features (see the feature list for explanation) we do not implement any of these in iteration 1, although doing design to plan for the future is always important, so we include these in our design documentation.

**Script Testing:** We propose the following approach on how a user may test their HTML5 application with a test script: After setting up the number of devices and which devices are connected to which networks, the user may access a GUI which has supplies the user with two lists: one of specific devices and one of events which can happen on these devices (eg incrementing a value, editing a picture, disconnecting from a network, joining a network, whatever). By pairing an item from the list of devices with an event, and then ordering these pairs, the user creates a timeline of network events which the server may execute. By doing this, the user is essentially building a test script without writing any actual code. The timeline/script could be saved and edited. Results could be saved in great detail with timestamps along with logs of what devices have what data so that the user could review them later

**HTML5 app deployment**: Simulation administrator and users may upload their HTML5 file to our server. They may choose to deploy this HTML5 application to specific users, to specific networks, or to the entire simulation. The users will then run these HTML5 applications.

**Modules**

Database: The database module contains all of the code required for interacting with the mongoDB database. This includes functions for saving, removing, and manipulating the database objects which represent device and simulation states.

Server: The server module is responsible for initially serving the web application to the client, as well as handling various requests for information provided by the client. The server facilitates communication between the database and client modules.

Client: The client module has various responsibilities. As such, we’ve broken it into a variety of sub-modules:

HTML5 rendering: The HTML5 rendering module is responsible for managing the appearance of our web app, as well as how it behaves when information changes.

Simulation Logic: This module manages the logic of the simulation, such as how networks are added.

Local Storage: The local storage module manages how simulation and user data is stored on a device.

Server Communication: The server communication module handles sending and receiving of data from the server.

GUI Rendering: All files related rendering the GUI on the client side

**Module Design Decisions**

We chose these modules as we believe that they cover the core aspects of our system. As well, these chosen modules are designed to be decoupled. Decoupling the modules allows for people to work separately on the different modules without worrying about dependencies with other modules. We believe that this will cause our software to be malleable and reusable for future iterations, and therefore provides good architecture.