CS4770 Project

Iteration 2
February 25, 2015

Group Members:
Noah Fleming
Emily Innes
Olanrewaju Okunlola
Ryan Martin
Andrew Caines
Mark Gauci

Table of Contents

Key Decisions - Iteration 2

Modular Approach

This iteration focuses on laying the groundwork for our system. This involves creating a server which can handle the creation of multiple simulations to which devices and networks belong. As this is the baseplate work for our system, which may need to be updated and changed in the future it is critical that we employ a modular design. To do this, we chose to employ such patterns as the Model View Controller (MVC) pattern in order to decouple our code and ensure that classes are disjoint. The client side functions as the “view”, the server side as the “controller” and the database as the “model”.

Server Client Independence

We aimed to decrease the chance of having contradictory calls to the server. To do this, we created an event queue and decreased the number of calls each client makes to the server. When the client performs a task, each event in that task is added to an event queue and packaged into a single JSON object which is sent as a single call to the server. The server, after receiving the object, unpacks this object and completes each event sequentially in queue (FIFO) fashion. This avoids having contradictory calls in the server. This has the added benefit of decoupling the server and client as much as possible and increasing cohesion. The server is responsible for server tasks, and the client is responsible for processing client tasks. One of the greatest benefits of handling server/client interaction in this form is that it allows the client to apply changes even while offline. These changes are then applied to the server when the device attaches to the server, as long as these changes do not conflict with any other previously handled events on the server. To avoid conflicts, events are executed sequentially based on the time in which they occurred.

Database

Although a database was not required for this iteration, we decided to include database support because we believe that a reliable form of data storage was essential to the boilerplate of our program and was tackled in iteration 1. As we have found the MUN servers to be rather unreliable we have chosen to host two databases which mirror one another on both the MUN and a separate server. This ensures that any loss in service from the MUN servers will not cause any of the data to be lost.

Token Propagation

We settled on what we believe to be both the cleanest, most reliable, and secure method of token propagation and verification. When each simulation is created, a token is created for each device which the simulation is allowed to have. These tokens are in the form of a unique

key generated by javascripts cryptographic module in order to ensure that they are a sufficiently difficult to guess. These keys are then sent via email to users who are allowed to join simulations. In future iterations, we will allow alternative methods of token propagation as listed below. After receiving a key, a user may then access the simulation and enter the key into the simulation. The key is then verified with the database of keys to check that it is valid. If it is valid the token is then stored in the local storage of that device on that browser, and the token is marked as used in the database in order to ensure that the token is not used more than once. Using local storage ensures that the token is device browser and device specific.

GUI Interaction

New to this iteration, end users are able to use a drag-and-drop style Graphical User Interface for modifying the network topology of a simulation. This is implemented so that adding, moving or removing devices across networks or partitions is far more user-friendly. As well, a user has the ability to view the history of a specific device with a mouse click. We used the power supplied from the Interact.js module to give us this added functionality, and this directly benefits the end user by simplifying the process of modifying a network topology. Currently, we have several UI layout options for the client to choose from and will be updating the look and feel of the existing GUI to not only better fit the wants and needs of the client, but also to improve the user experience when using the GUI.

Logging and Timestamps

A necessary feature to any project is the the aspect of logging, and this one is no exception. We use Express-Logger found in the Node Package Manager which automatically logs developer-level events that allow us (the developers) to open the log file after running the application and view the list of events that occurred at run time, but more importantly any errors the project encountered. In turn, this makes debugging easier for us given there are errors. We also capture a timestamp of events so that an end user can view the history of a device with respect to a running simulation. They do this by navigating to the topology GUI, selecting a device and viewing its list of completed actions.

Code Refactoring

With this iteration, we were given a network simulation interface with which our pre-existing code must work. In order for the two code entities to work together, we needed to refactor our written code to allow for the integration of the given interface. In doing so, we continually followed proper coding principles and improved upon the existing code cohesion and reduced any unnecessary coupling between modules within our project. The code is now more concise, clean and clear to read and understand.

Project Design Decisions

Programming Languages – HTML5, CSS3 through Bootstrap, Javascript through Node.js and Express

HTML5, CSS3, and Javascript were selected due to their ease of use and their flexibility across multiple platforms. With so many new devices and mobile operating systems appearing on the market, it makes sense to code once, distribute once as opposed to using several different development environments and the cost that sometimes comes with doing so. Using these three languages, the web app is compatible with all browsers and mobile devices, allowing this project to reach the most clients for the least cost. Using Node.js allows us to quickly build and deploy our application. Bootstrap allowed us to apply our cascading style sheets quickly and efficiently.

The only downside to this decision is that anyone with older browsers and devices may be unable to use the software. This concern is fairly minimal as browsers and mobile operating systems are free to download/update and cell phones are normally replaced within a two year cycle or less.

Database - MongoDB

Achieving persistence of data was a goal for this iteration and to do so, we needed a database. We wanted to ensure we were able to store and call our tokens and partition information. MongoDB integrates seamlessly with Node.js and Express. Being a NoSQL database certainly presents its challenges since many of us had never worked with it before, however its simplicity and speed quickly proved to be beneficial.

Development Environment - Eclipse

Eclipse is an open-source, multi-language IDE that offers a variety of extensions that can be used for any development situation. Given the high cost of some other IDE's, for the purposes of this project and keeping costs low, Eclipse will give us the tools to develop this project.

Version Control – Git

Git has grown in recent years as the go to source for version control. It is open source, available on all platforms, and can be used by both enthusiasts accessing GitHub, or in our case used in the enterprise environment, allowing our team members to work collaboratively with minimal time wasted.

Browser – Chrome

Chrome has quickly become the go to browser for the modern age. It works equally well on all platforms, acts as its own operating system, offers a variety of extensions and is used by the most users. This is why it is imperative that our application work well within the Chrome browser first, and adjustments made to accommodate other browsers after the fact.

Code – Module Based

By breaking up the code into modules, it ensures that we adhere to proper object oriented coding practices. This ensures the code is manageable, loosely coupled and able to respond to change with minimal effort and time. As the project grows, we can add needed complexity without having to completely rewrite the entire project. This makes for good housekeeping as the resulting code is neat, tidy and allows for our team to test the code thoroughly throughout each iteration. This ensures we have a well-established foundation with which to continue building our project.

The code will be commented throughout to ensure that anyone new who joins the project will be able to see what the code is doing, and quickly catch up with the rest of the team.

Use Case, Risk and Features Key Decisions

When creating the use cases for this project, we had to look at what we wanted to accomplish from both the administrator perspective and the device perspective. Both the administrator and the device can create and destroy the networks they create. The administrator specifies at the start of the simulation what networks it wishes to create, a fairly standard use case. The device may need to create a peer to peer network with other devices when the device is unable to reach the server. This allows devices to continue communicating between each other, with all devices synchronizing with the server when communication allows.

The number of devices able to access an available network is determined by the number of tokens created when the network is formed. Each token is unique and provides specific identification for each device on the network. This ensures that only devices registered with the application may communicate.

The tokens can be distributed in a number of ways, such as by email, by text or by registering on the website itself and receiving it online. These tokens are unique IDs generated within a data base. Code exists to ensure that no more than the specified number of tokens is created such that when the number of tokens desired equals the number of records in the database, then no more tokens can be created and the user requesting the token is told no token is available.

Test scripts are used for unit and integration testing. They also allow for activity tracking such as what device is joining what network, at what date/time and at what location,

error tracking, letting the moderator know what part of the site may have failed or what a user has done wrong. These scripts can be automatically active in the background while the application is running, with little input from any user or with any UI involved. Said scripts would generate log files in a text format to be retrieved by the moderator on a daily basis or as emergencies dictate.
Each device that wishes to join the application must register within the simulation for identification and routing purposes. Multiple simulations running concurrently is possible, so the server must be able to respond in kind. To avoid any needless extra code, the partitions of the virtual networks would be setup at the launch of the initial simulation, with the hopes that eventually each of the networks would be able to communicate with each other, achieving a consistency that can be modeled.

Virtual devices can connect to other devices within the simulation, depending upon which networks they have access to. Once network access has been established, the user would then be able to distribute files across the network to each device.

For a more technical description of the features and risk decision making process, please see Feature List Justification.

Sequence Diagrams

When creating the sequence diagrams for our project, we looked at the fundamental key features of our system. The sequence diagrams represent the core foundation of this system and significantly reduce misunderstanding of what key features are and how they will eventually be implemented. These diagrams are primarily used to show essential interactions between core objects in sequential order.

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.

Use Cases

1. The user wishes to create the simulation and specify general parameters such as the number of devices that will run on the simulation and number of virtual networks.

2. The user wishes to enter and/or edit the number of devices the simulation will be running on.
    a. A selection mechanism needs to be available to allow the user to select the number of devices in the simulation.

3. The user wishes to enter and/or edit the number of virtual networks in the simulation environment. The user may wish to simulate the data replication across several virtual networks. We need to give the user an option to create virtual networks.

4. The user wishes to register a device for a particular virtual simulation session.
    a. A token based system of registration can be used. The virtual simulation is assigned.
    b. A fixed number of tokens corresponding to the number of devices in the simulation environment. Each device needs to be assigned to at least one virtual network set in the simulation and consumes a token provided by the virtual simulation. Once a token is consumed it cannot be used by other devices.

5. The user wishes to deploy an HTML5 application across all devices in our general simulation. Applications may make use of replicated data types.
    a. It would be useful for the user to have some visual representation of how data is being moved through the network for devices in a partition. A small feature should allow the user to simulate deploying the application on all devices.

6. The user wishes to specify which virtual networks can communicate with each other and which ones cannot. E.g. the mobile users at a hiking trail may not be able to talk to the airport mobile users network due to the lack of a network connection between them. The user needs an option list once they have created the device and virtual network set to allocate which virtual networks can communicate with the other.

7. The user wishes to eventually have all virtual networks be able to communicate with the others in order to model the purpose of the simulation - eventual consistency.
    a. Ideally a mechanism would allow the partitions to be capable of communicating with each other by sending replicated data types. In this case the user should be able to visually see consistency across all devices running the application in

our system as data becomes available to all devices on the network. All the applications should have consistent data types and states.

8. The user wishes to direct data flow on the HTML5 application across all available virtual networks. The user should be able to test the replication of data types across the application

9. The user wishes to move a device from one virtual network to another, delete a device from a virtual network (bring it offline) or add it to a network.

10. The user creates a virtual network for a device and allows other devices to join this new virtual network. This means the modules for managing virtual networks needs to be decoupled from mobile devices. A mobile device can both manage its own self created virtual network and be managed by an admin portal for the simulation environment.

11. The virtual device user moves or removes virtual devices from its own self-created virtual network. Any virtual networks created by this device can also be destroyed.

12. The user should be able to run some test scripts for testing the simulation system(s) in units and as a whole. These scripts can be used to automatically run the simulation environment without user input or ensure each unit works as expected.

13. The user checks error tracking logs for the system. Example of logs includes - the server taking long to respond to requests, virtual devices dropping out of virtual network range, networks going down. The logs should include the timestamps .

14. The user checks activity logs for the system. Examples of activity logs includes - device id, date/time stamp, GPS location, network membership, P2P activity.

15. The user deploys the simulation application on several browsers (Chrome, Safari, Opera, Firefox and Internet Explorer). The environment should be consistent in display and functionality across all browsers.

16. The user displays and manipulates the network topology. This includes joining/connecting networks, separating networks, deleting networks, adding networks.

17. The user displays and manipulates the device topology. This includes adding and removing devices from a network.

18. The user selects a device to see pertinent device information including, but not limited to the device type and the token used.

19. The user is able to display the network and device topology from a previous state by searching for or selecting a date/time stamp.

Breakdown of Use Case:

Use Case 16: Display and manipulate the network topology.

Justification for choosing this use case:
This use case is a new feature introduced by the client. It allows the user to see a graphical representation of the networks within a simulation. The user can see how each network is or is not connected and can manipulate these networks to suit their network connectivity needs.

Use Case: Display and manipulate the network topology.
Primary Actor: The manager of the simulation
Scope: The Simulation environment
Description: The user wishes to display the simulation in a graphical format that is interactive, allowing them to effect network change according to their needs.
Preconditions: The server running simulations is running and able to handle requests. While the simulation is running, at least one or more networks and one or more devices must be currently active to effectively display this utility.
Basic Flow:

1. The system provided a network and device topology page, displaying all active networks, devices and any relevant connections between them.
2. The user is able to interact with the current network and connection objects.
3. The user alters the available networks as needed by creating or deleting additional networks.

Alternate Flows:
3: The user does not have the rights to manipulate the topology
3.1 The network topology is available for viewing, though no changes can be made.

4: The user connects the wrong networks
4.1 The user selects the connection between the networks and deletes the connection.
4.2 The user then creates a new connection between the two or more desired networks.

Postcondition:
The network topology is displayed correctly according to the users needs.

Use Case 17: Display and manipulate the device topology.

<u>Justification for choosing this use case:</u>
This use case is a new feature introduced by the client. It allows the user to see a graphical representation of the devices within a simulation. The user can see where each device is located, either in a specific network or in no network at all and can manipulate these devices to suit their device connectivity needs.

Use Case: Display and manipulate the device topology
Primary Actor: The manager of the simulation
Scope: The Simulation environment
Description: The user wishes to display the simulation in a graphical format that is interactive, allowing them to change a devices location according to their needs.

Preconditions: The server running simulations is running and able to handle requests. While the simulation is running, at least one or more networks and one or more devices must be currently active to effectively display this utility.

Basic Flow:

1. The system provided a network and device topology page, displaying all active networks, devices and any relevant connections between them.
2. The user is able to interact with the current device objects.
3. The user clicks and drags the available devices their desired destination as required.

Alternate Flows:

5: The user does not have the rights to manipulate the topology
5.1 The network topology is available for viewing, though no changes can be made.

6: The user clicks and drags the current device to the wrong network
6.1 The user selects and drags the device to the desired network.

Postcondition:
The network topology is displayed correctly according to the users needs, with the proper devices shown within the desired networks.

Use Case 18: Display device information.

Justification for choosing this use case:
This use case is a new feature introduced by the client. It allows the user to view more detailed information on the device being selected.

Use Case: Display device information
Primary Actor: The manager of the simulation
Scope: The Simulation environment
Description: The user wishes to display the currently selected device's information, up to but not limited to it's device type and token information.

Preconditions: The server running simulations is running and able to handle requests. While the simulation is running, at least one or more networks and one or more devices must be currently active to effectively display this utility.

Basic Flow:

1. The system provided a network and device topology page, displaying all active networks, devices and any relevant connections between them.
2. The user is able to interact with the current device objects.
3. The user clicks one of the devices.
4. The desired information dealing specifically with that device is displayed on screen.
5. When the users curiosity is satisfied, the user closes the device information window, returning to the network and device topology screen.

Alternate Flows:

7: The user does not have the rights to view the details on the device in question
7.1 The network topology is available for viewing, though no device information is available.

Postcondition:
The device information is displayed correctly, and upon closing the device information window, the network and device topology screen is correctly displayed.

Use Case 19: Display network and device topography from a previous state.

Justification for choosing this use case:
This use case is a new feature introduced by the client. It allows the user to recall a previous state of the topology, showing a graphical representation of the network from the past.

Use Case: Display network and device topography from a previous state
Primary Actor: The manager of the simulation
Scope: The Simulation environment

Description: The user wishes to recall a previously saved network topology state view it on screen.

Preconditions: The server running simulations is running and able to handle requests. While the simulation is running, at least one or more networks and one or more devices must have been created and manipulated to create the required log files and the changes to show the difference between past and present states.

Basic Flow:

1. The system has logged the creation of and changes to the currently active network topology.
2. The user recalls the saved topology by selecting the <<insert button name here>> and entering the date/time information for the desired state.
3. The user selects OK.
4. The previously saved topology is displayed for the user on screen.
5. When the users curiosity is satisfied, the user closes the window, returning to the current network topology screen.

Alternate Flows:

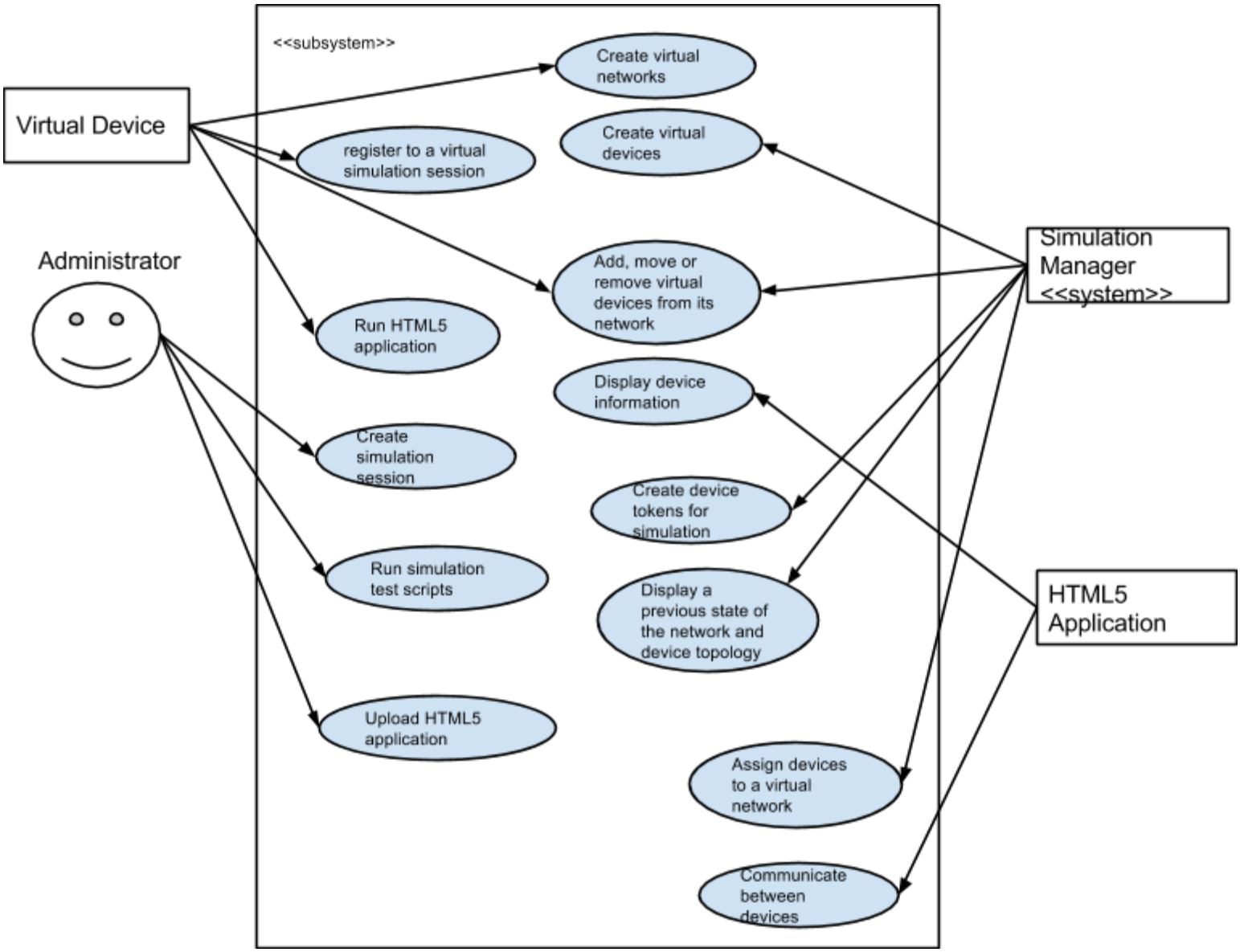
8: The user does not have the rights to recall previous network topology states
8.1 The <<insert button name here>> for displaying the window that allows for date/time searching is not displayed.

9: The user enters the wrong date/time stamp.
9.1: The user closes the window displaying the incorrect results.
9.2: The user then re-enters the desired date/time stamp and acquires the desired results.

10: The user enters the right date/time stamp, but sees no previously saved state of the topography.
10.1: The user notifies the administrator of the deficiency and repeats the process later.

Postcondition:
The device information is displayed correctly, and upon closing the device information window, the network and device topology screen is correctly displayed.

Use Case Diagram



Traceability Matrix

Feature											
Use Cases	Feature 1	Feature 2	Feature 3	Feature 4	Feature 5	Feature 6	Feature 7	Feature 8	Feature 9	Feature 10	Feature 11
Use case 1	✶		✶								
Use case 2	✶			✶							
Use case 3	✶		✶								
Use case 4	✶	✶									
Use case 5	✶				✶						
Use case 6	✶		✶								
Use case 7	✶		✶			✶					
Use case 8	✶			✶		✶					
Use case 9	✶			✶							
Use case 10	✶		✶	✶							
Use case 11	✶		✶	✶							
Use case 12	✶					✶	✶	✶			
Use case	✶									✶	

13											
Use case 14	✶										✶
Use case 15	✶								✶		

Features List

1. Process incoming page requests and provide correct responses
2. Possesses a robust token system in order to selectively allow devices to participate in a simulation
3. Token propagation to different users via Email
4. Create and manage multiple virtual networks with:
    a. Ability to add, move, or remove any virtual devices from any networks
    b. Ability to allow or deny communication between members of different networks
5. Create a virtual mobile device in the browser which:
    a. Can register itself with a network simulation session
    b. Can connect and disconnect from pre-existing virtual networks within its simulation
    c. Can create, manage, and destroy virtual networks which it has created
    d. Can run HTML5 applications which can communicate with other virtual devices running the same application
6. Register HTML5 applications with a simulation session which any participating virtual device may run
7. Test the performance of the HTML5 applications
8. Automatic execution of test scripts
9. Track and manage multiple network simulation sessions
10. Compatible with all browsers
11. Token propagation via multiple methods, which the administrator may select from.
12. Error logging for the simulation environment available to the moderator
13. Have activity logging for the HTML5 application, tracking user login date, time, location and network activity.

Feature List Justification

We present justification behind the ordering of the features in the feature list. This includes

explanations of why these features are risky and their applications to the three Q's or architecture.

1. Server features such as handling incoming page requests and providing correct responses is a very risky feature to our architecture. It is the foundation of our program, as it handles channeling data to and from users, therefore it is essential to our system. Only a single member of our team is familiar with node-js which we will be using to implement the server, therefore we are unsure of how to implement it. Finally, we are not entirely sure what the server should do and how to handle it. Therefore this satisfies the three Q's of architecture, and this feature is very risky and should be handled early by architecture.
2. Having a robust token system is critical to our system, it allows for devices to access a simulation, and without it none of the other required features for the system would be possible to implement. Therefore this is an essential feature to our system. This is a challenging feature, as actual devices as well as simulated devices must be able to have tokens, the tokens must be unique, and can only be used once. Therefore, we do not entirely understand what this means and how this should be implemented. Thus, this feature satisfies the 3 Q's of architecture and is at high risk to our design, so we should tackle it with architecture as early as possible.
3. It is necessary for our tokens to be propagated to devices which should be added to our system. The most straight-forward, sought after, and secure way we believe to do this is with email. As token propagation is required for any device to join a network, it is essential to our system. We understand this feature, but are unsure of entirely how to implement this feature. Therefore this satisfies 2 out of 3 of the three Q's of architecture, but as it is such a central feature to our system, we place it high on our features list.
4. Create and manage multiple virtual networks. This is a very essential feature to our system, as it is the part of the core of our program and most of the other features would be nonexistent or useless without it. We believe that we do not understand this feature in its entirety. We do not know how the networks will have to behave in the future, and what sort of communication will be required. As we do not entirely understand this feature, we do not understand how to implement it. Therefore this feature satisfies the 3 Q's of architecture and is very risky. Thus, we handle it with architecture early on and place it high on the priority of the feature list.
5. Create virtual mobile devices and allow connection of real mobile devices to system. This is essential to our system, but we believe we understand how to handle the implementation and understand how it should work. This satisfies one of the three Q's of architecture, and therefore is not as risky as the ones above but should be still handled by architecture and as early as possible, as it is one of the most essential parts of our system.
6. Test the performance of given HTML5 applications. This is essential to our system as it is in the given vision statement of the project. We are unsure of what it means, as we do not know how the system should interact with HTML5 applications and what should be done. As well, we do not know how to implement this, as we do not know how

these applications are to be sent or run. Therefore, this satisfies the three Q's of architecture, and this feature is very risky and should be handled early by architecture.

7. Automatic execution of test scripts. This is essential to our system as it is given in the vision statement of the project. We are not exactly sure what this means, as we do not know what scripts the customer would want run and how to execute them. Therefore we are also unsure of how to implement. This satisfies the three Q's of architecture and is therefore a very risky feature and should be handled by architecture early on.
8. Register HTML5 applications with a simulation session which any participating virtual device may run. This feature is essential to our system as it is stated in the vision document of the project and therefore must be included in the program. We are unsure of exactly how to implement this and how this should be done. Therefore this satisfies the three Q's or architecture, and is quite risky. Therefore we handle this with architecture.
9. Track and manage multiple network simulations is essential to our system, as it is one of the required features to the system, although we do not believe that it is of utmost importance and therefore it is significantly low on the features list. We believe that we understand what this concept means, but we are unsure of how this should be handled and how to implement it, especially along the lines of how various simulations should be accessed. Therefore it satisfies two of the 3 Q's of architecture, and should be handled by architecture and design. For this feature, our belief is that it would be simplest to have a website which allows you to select the simulation you wish to connect to and then route you to that specific simulation.
10. Token propagation via multiple methods this is not essential to our system, although it would provide added flexibility and allow certain users not willing to supply their email addresses, or without an email address to access our system. It would as well allow for administrators to handle different kinds of simulations with more or less security (for instance, a user being assigned a token when accessing the website is much less secure than email authentication). We are unsure of how to handle this, but we understand the feature. Therefore this satisfies 1 of the three Q's of architecture, and so is low on our list of features, although we still believe it is an important feature.
11. Compatible with all browsers. This feature is not very risky. It is not "essential to our system", we know how to handle it, and we know how to implement it. Therefore it is not pertinent to include architecture for this feature.
12. Error logging for the simulation does not satisfy any of the 3 Q's of architecture, so it is low on the features list. This feature will allow for much simpler debugging and programming of our system and therefore would streamline the design process. Therefore, we believe that this is a necessary feature to our system.
13. Activity logging for the system would allow the administrator and possibly users to view activity of HTML devices and the simulation. This does not satisfy any of the 3 Q's of architecture, so it is at the bottom of the features list. We believe that this feature would be very useful for understanding the effect that certain HTML5 applications and scripts have on the simulation and provide us with much more in depth information, which could be very useful for tracking the effects of 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.

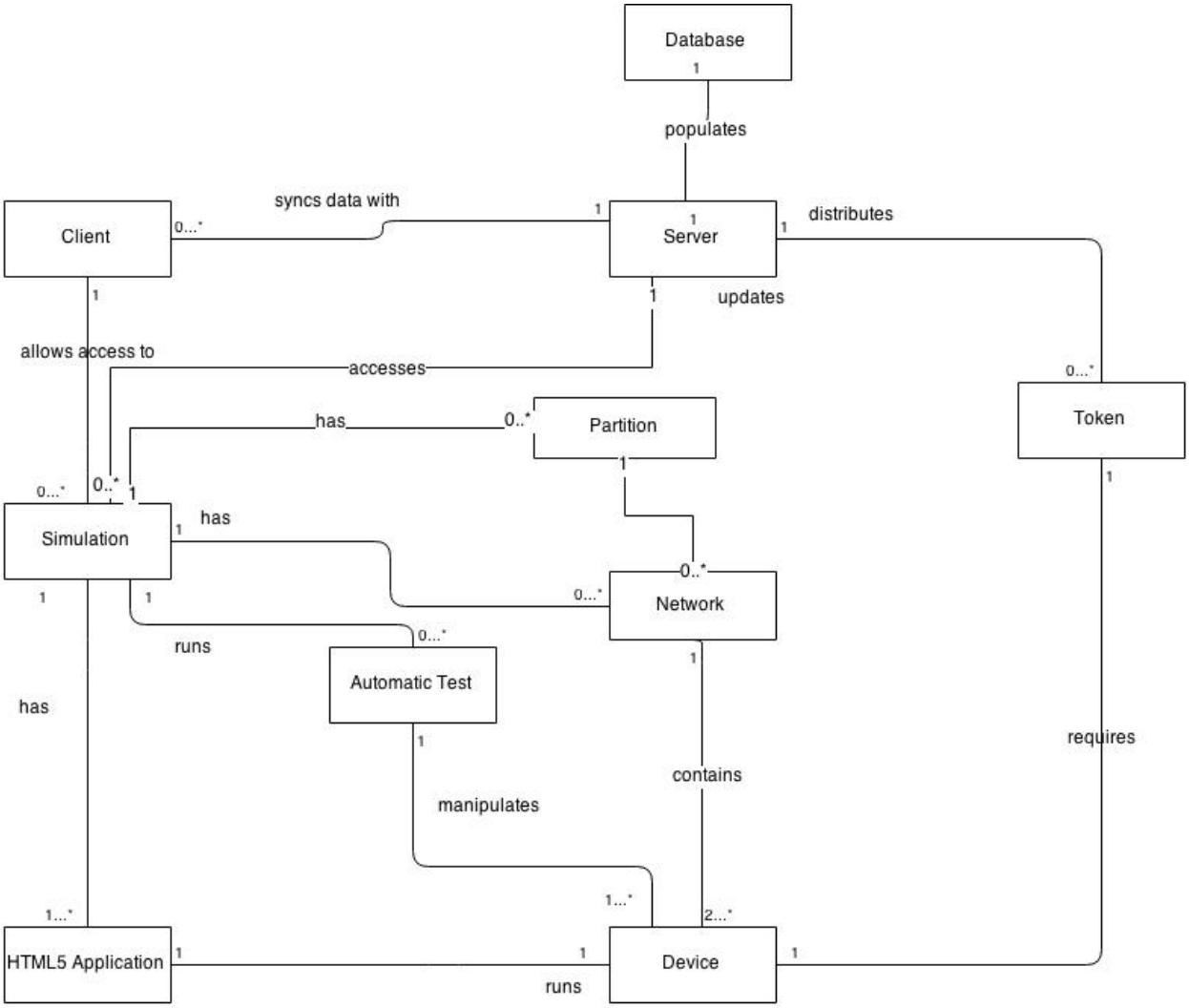
Server Communication: The server communication module handles sending and receiving of data from the server.

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.

Domain Model



Domain Model Design Decisions

We designed the domain model in order to give a concrete idea of the conceptual classes for our system. This is the foundations of our design, from which we can construct the modules and hypothesize concrete classes. We chose to put the HTML5 application as a conceptual class as it allows us to see the interaction between virtual devices and the simulation. This allows us to plan for how the HTML5 applications should be run on devices on the network and how this should interact with the simulation. Thus, we believed that this was an appropriate conceptual class. We include Automatic Tests as their own conceptual class, as we believe it is important to plan for the interaction with the simulation and virtual devices. Adding this as a conceptual class allows us to view the associations between these classes. The remaining conceptual classes followed logically from the vision statement of the software.

Sequence Diagrams

We chose the following sequence diagrams as they represent the key features from the feature list which we believed were the most essential to our system, risky, or not understood. The justification behind why these features are risky can be found under the features list. We believe that these sequence diagrams create a foundation for how to develop our software, and significantly reduce the risk posed by not understanding what certain features are and how to implement them.

The getEvent from client sequence is central to our server connecting between the client device. It handles getting information from the client, handling that information, and returning the updated state of the program to the client. Thus we handle this in sequence diagram 1.

One of the events handled by the simulation in sequence diagram 1 is the creation of a simulation. This is detailed in sequence diagram 2, in which information from the client is processed in order to create a new simulation.

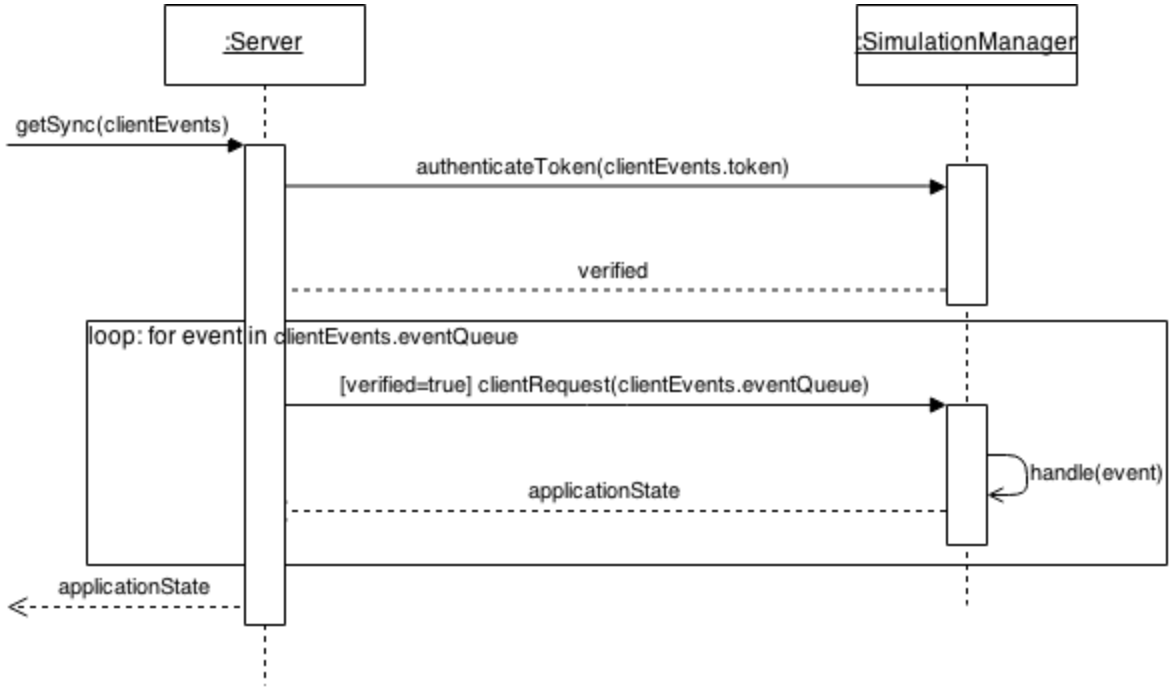
Sequence diagrams 3 details another event handled by the simulation in sequence diagram 1, which is the authentication of a simulation.

Tokens are critical to the security and management of our server; therefore, understanding how unique tokens are generated and handled is essential to our system. We detail this process in sequence diagram 4.

Our system is required to be able to run HTML5 applications, scripts, and simply send data between servers. Therefore, it is essential to plan for how data is sent between devices on a network and how to distribute information to all things on a system. Ultimately, we created sequence diagrams 5 and 6.

Both from the GUI interface introduced in Iteration 2 and for all previous Iterations, our project allowed for the creation of networks and partitions. This is a crucial and core feature our project must be able to do, and Sequence diagrams 7 and 8 outline the specifics of these two procedures from the perspective of the server side.

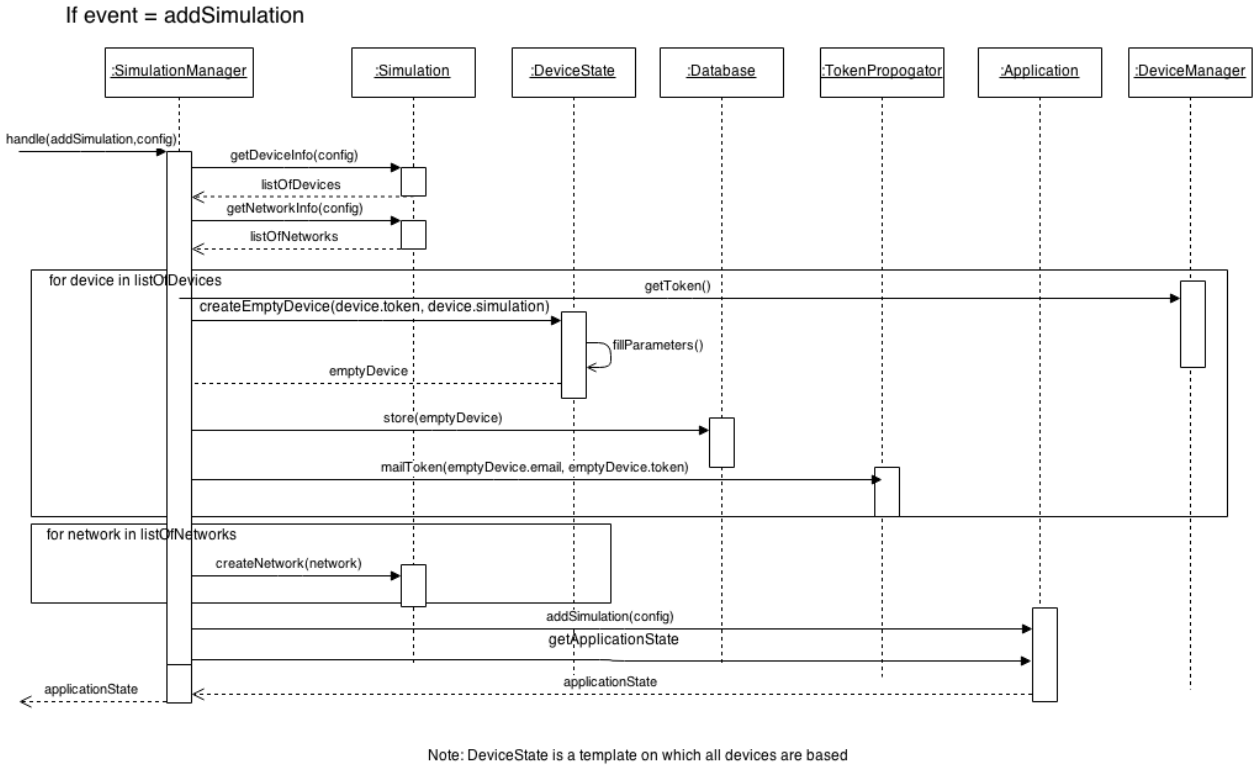
Sequence Diagram 1: Get Events From Client



This sequence diagram details the server receiving a block of events from the client. First, the token of the user is authenticated to ensure that this is indeed a valid device to be receiving information from. The “clientEvents” which is just a block of individual events which the client has computed. This object is then parsed and each event is handled sequentially, ensuring no overlap. The events are handled based on what kind of event they are in the following sequence diagrams. Finally, the state of the application is returned to the device which will then be updated accordingly. This is necessary, as the device may be an admin in which case it must be able to view different simulations. As well, a device may be a device in multiple simulations.

As receiving information from the client is the core functionality to our system, creating a sequence diagram was essential for the planning of how this should be accomplished.

Sequence Diagram 2: Create A New Simulation



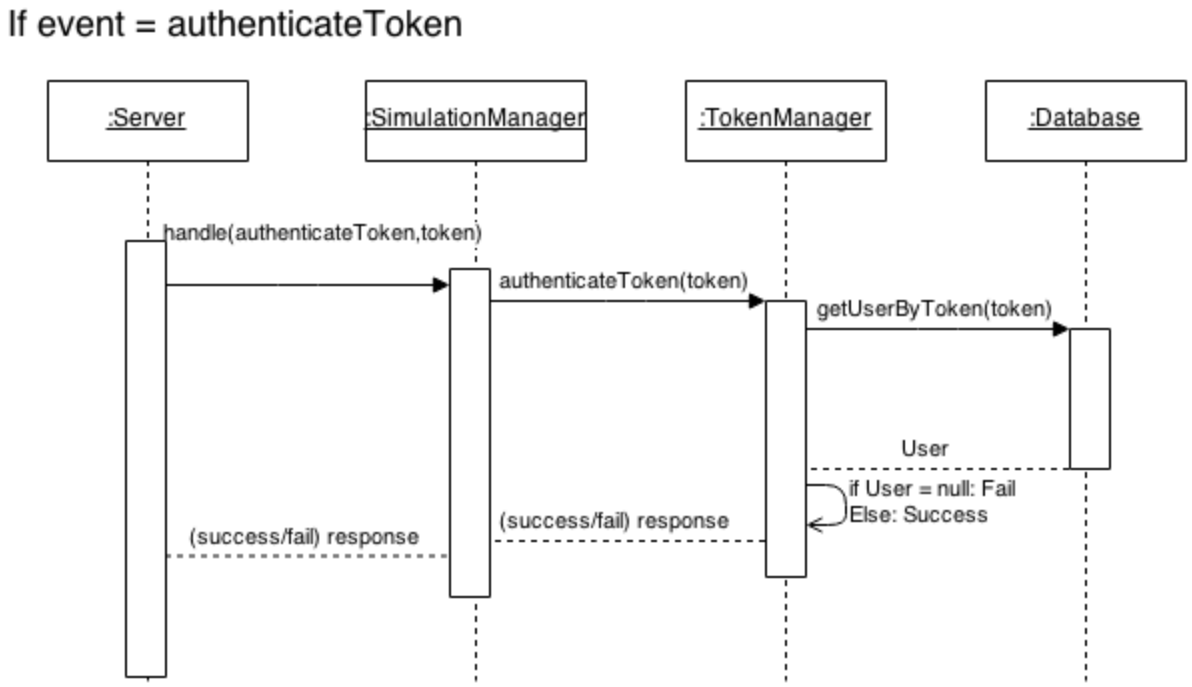
Note: DeviceState is a template on which all devices are based

This sequence diagram details the sequence of calls made when the creation of a new simulation occurs. The system is passed the number of networks and device tokens to create, as well as the method by which they would like to distribute the tokens all contained within “config”. We have outlined how this distribution would work for Email, and detailed possible other token distribution methods in the section Token Management. After these parameters have been supplied, the networks and tokens are created, and the tokens are added to the database holding all tokens. This database is used to keep track of tokens in use and ensure that tokens are unique. TokenPropogator may be replaced by “emailTokenPropagator”, or any other form of token propagation, which allows the tokenPropagatorStrategy to adhere to the strategy design pattern.

We believed that this was a necessary sequence diagram to create because it outlines arguably the most essential step in our system, the creation of virtual devices and virtual networks and setting up the simulation. We were unsure of exactly what it should do, and how to handle it.

Therefore, this satisfied all three Q's of architecture, and it was necessary to complete design for it. We decided to create the NetworkManager and TokenManager classes in order to ensure that every class was cohesive, took care of exactly one class, and decoupling of all classes. This was in order to make our software more streamlined and decrease dependencies between different classes.

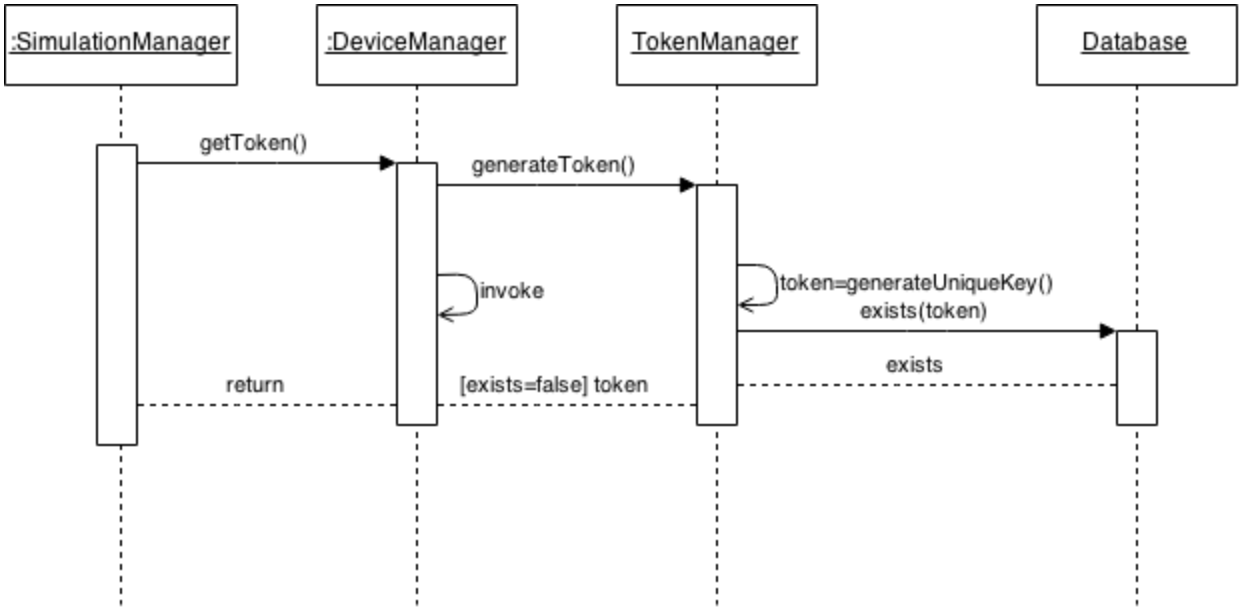
Sequence Diagram 3: Authenticate Token



This sequence diagram details the authentication of a token sent from the server. This follows from “handle(event)” in sequence diagram 1 in the case that the event is “authenticateToken”. The token object is delegated from the simulation manager to the token manager. This ensures that each class has exactly one responsibility and increases cohesion and decoupling. The TokenManager then checks with the database to see if a “user” with that token exists, as dummy users are created upon simulation creation with tokens attached to them. It then checks if the returned user is null or not and returns to the client the success of the authentication of the token.

We believed it necessary to design for this as token authentication is central to the security of our server. Therefore having a concrete understanding of how it should work is necessary in order to ensure that everything functions correctly, and that our design is modular.

Sequence Diagram 4: Generate Token

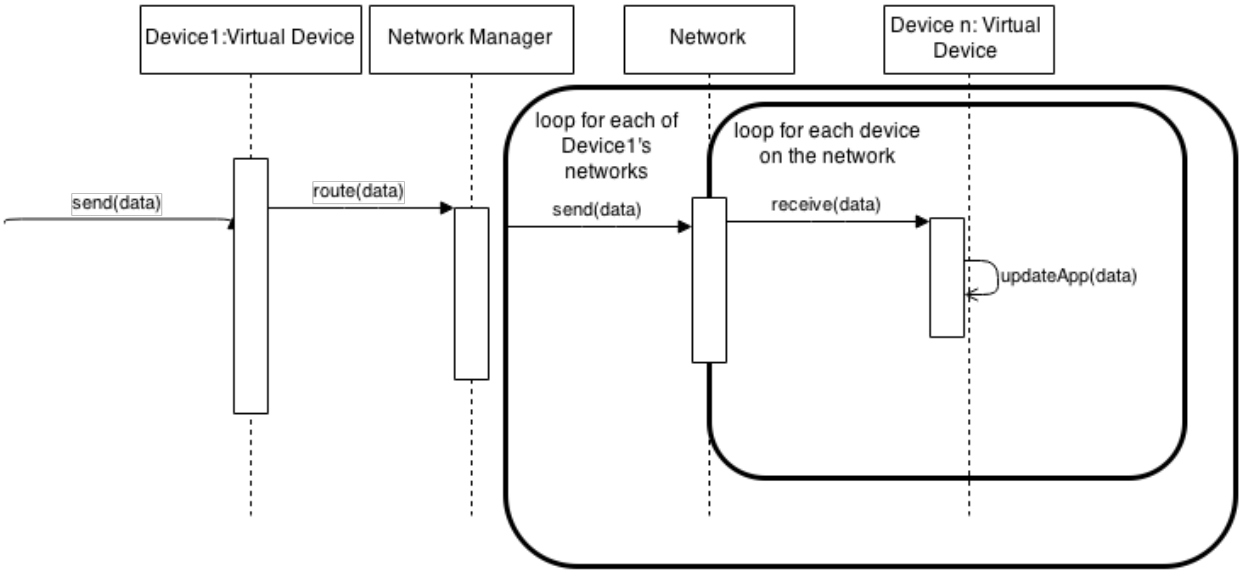


This sequence diagram outlines the generation of a unique token. This sequence diagram follows from sequence diagram 2. The request for a new token is delegated to the TokenManager in order to ensure that each class has exactly one responsibility and maximize cohesion as much as possible. The token manager generates a random string which is then checked with the database in order to ensure that the token is unique. The token is then returned to the simulation manager to be attached to that “dummy” user (a user without a device registered to it).

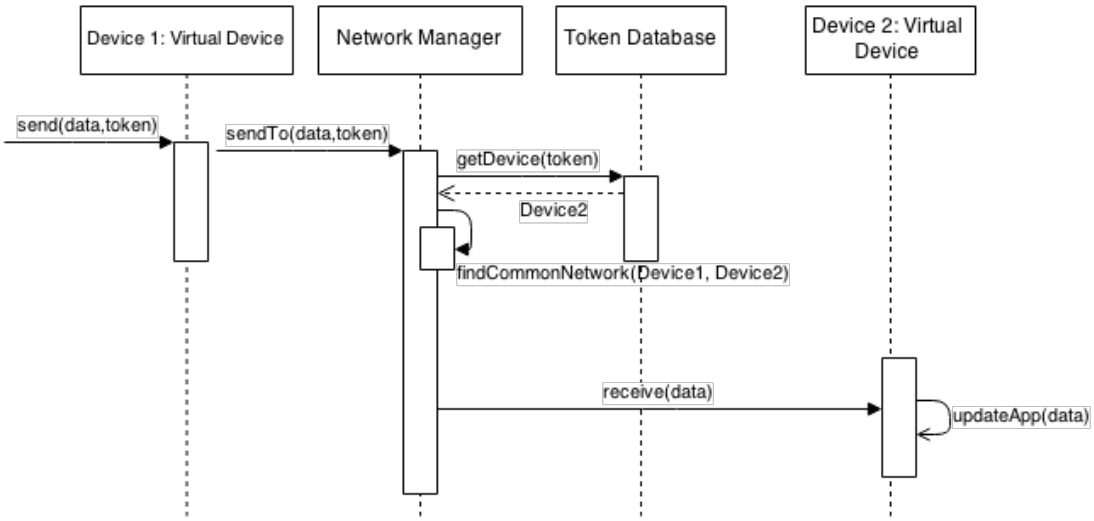
We did design for this sequence of events in order to ensure that the correct delegation throughout our system is accomplished. As well, since tokens are a core feature of our system, understanding their creation is an important feature to examine. We believed that sequence diagrams give us the best insight into this.

Sequence Diagrams 5 & 6

Case 1: Sharing data with all connected devices

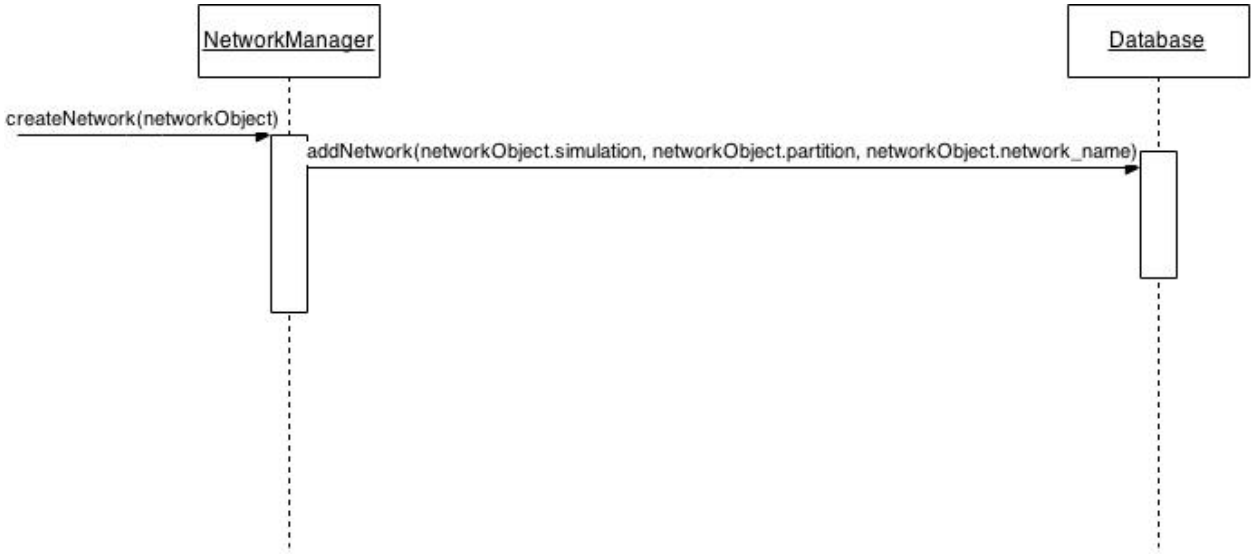


Case 2: Sharing Data with a specific device



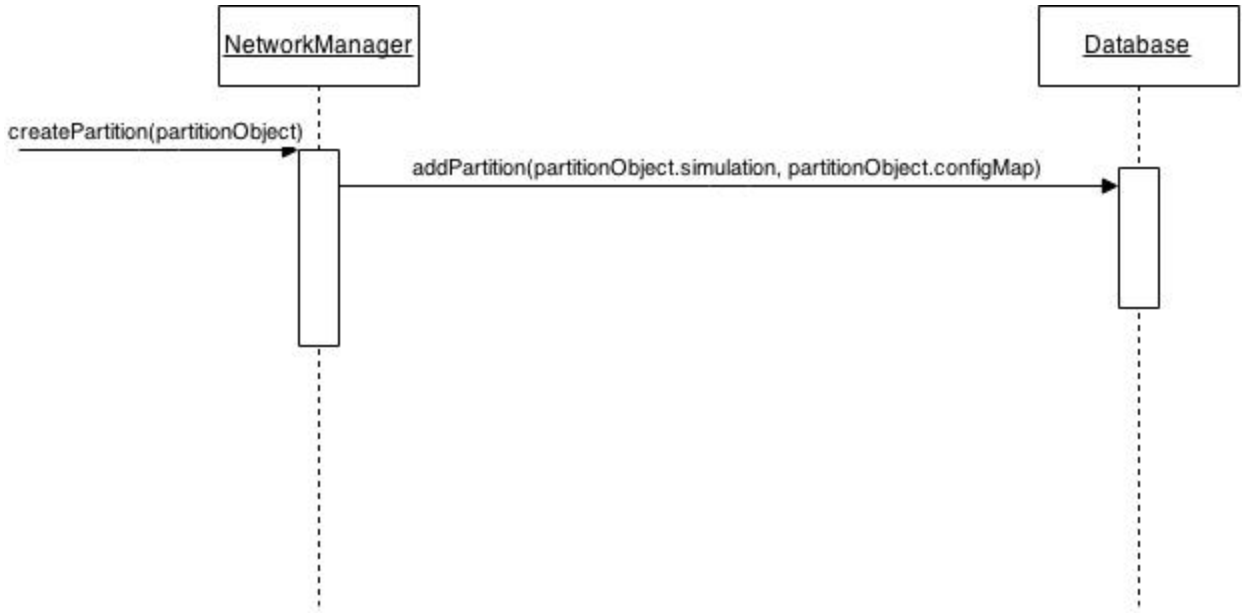
These two sequence diagrams detail information being sent to either a specific machine, or all machines connected to a network. This data could be a test script, an HTML application or anything else. We designed this sequence diagram to be completely data-general in order to handle whatever we may want to pass through the system in the future, allowing our architecture to be malleable and refactorable for future iterations.

Sequence diagram 7: Create Network



Here, the NetworkManager module within the server side code of the project receives a function call from another module with the argument of a network object. Next, the NetworkManager delegates the method call to a database object which then stores the network name, partition and simulation information in the database. This allows for persistency of network as well as other crucial information.

Sequence diagram 8: Create Partition



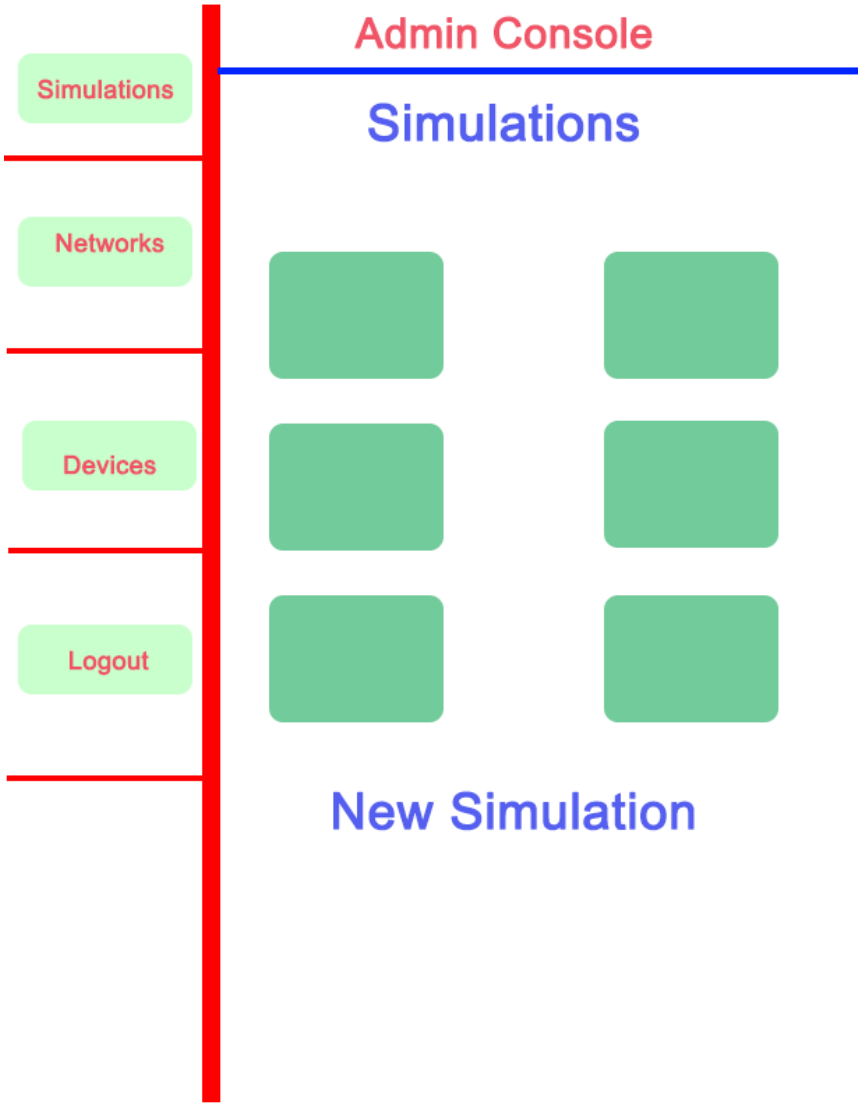
Similar to the above sequence diagram, the create partition method involves the interaction of the same modules (NetworkManager and Database), and stores the necessary partition

information to the database to allow for consistency. The difference between the previous sequence diagram being that the CreatePartition method is called with a partition object and delegates the operation to a database module.

Gui Sketches

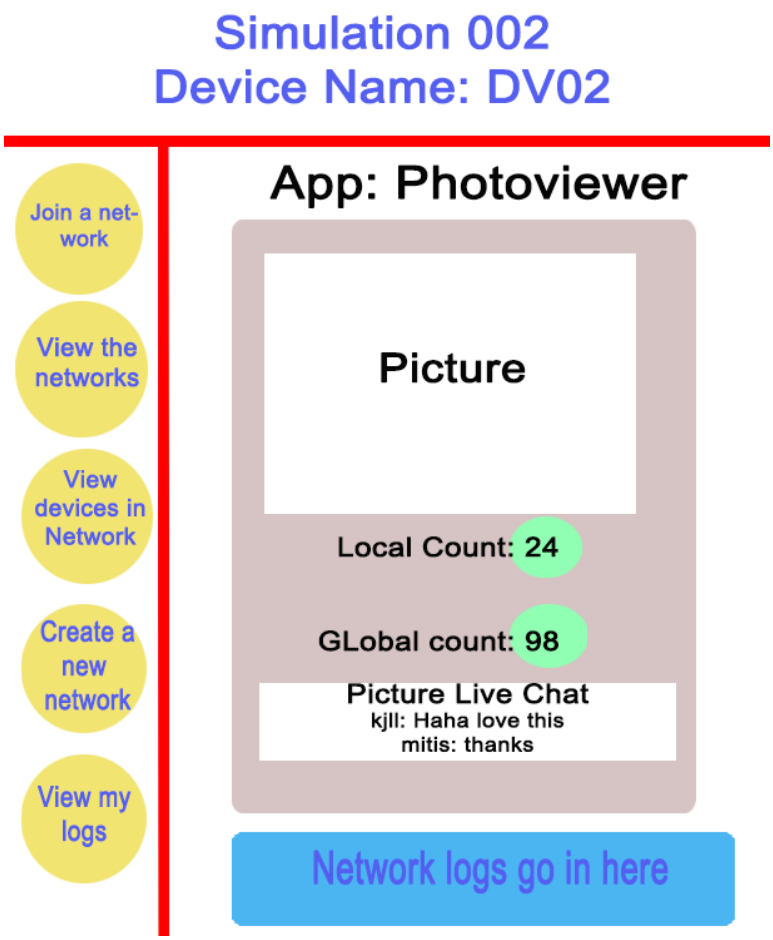
In order to plan for the appearance and functionality of our system, it is important to visualize how such a system would look. Doing this before our system is actually created allows us to minimize risk, and plan for the interaction between the user and the software. Therefore we created the following sketches of how the user interface should look and feel.

Admin Page



This sketch details the layout of an administrator of the application's page. Here, the green boxes indicate various simulations in the application. The "new simulation" button allows for the creation of new simulations. Various functionality is laid out within the left bar. The "networks" and "devices" tab allows a user to view all devices connected to all simulations, and all networks within all simulations.

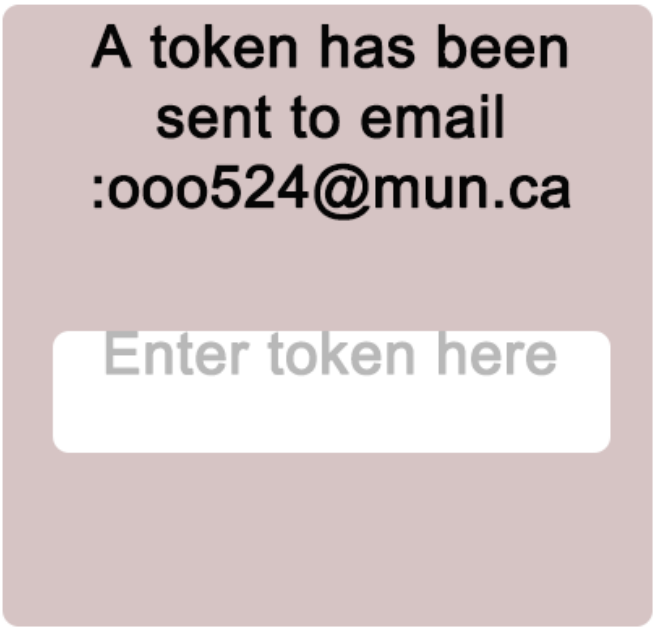
User Device View



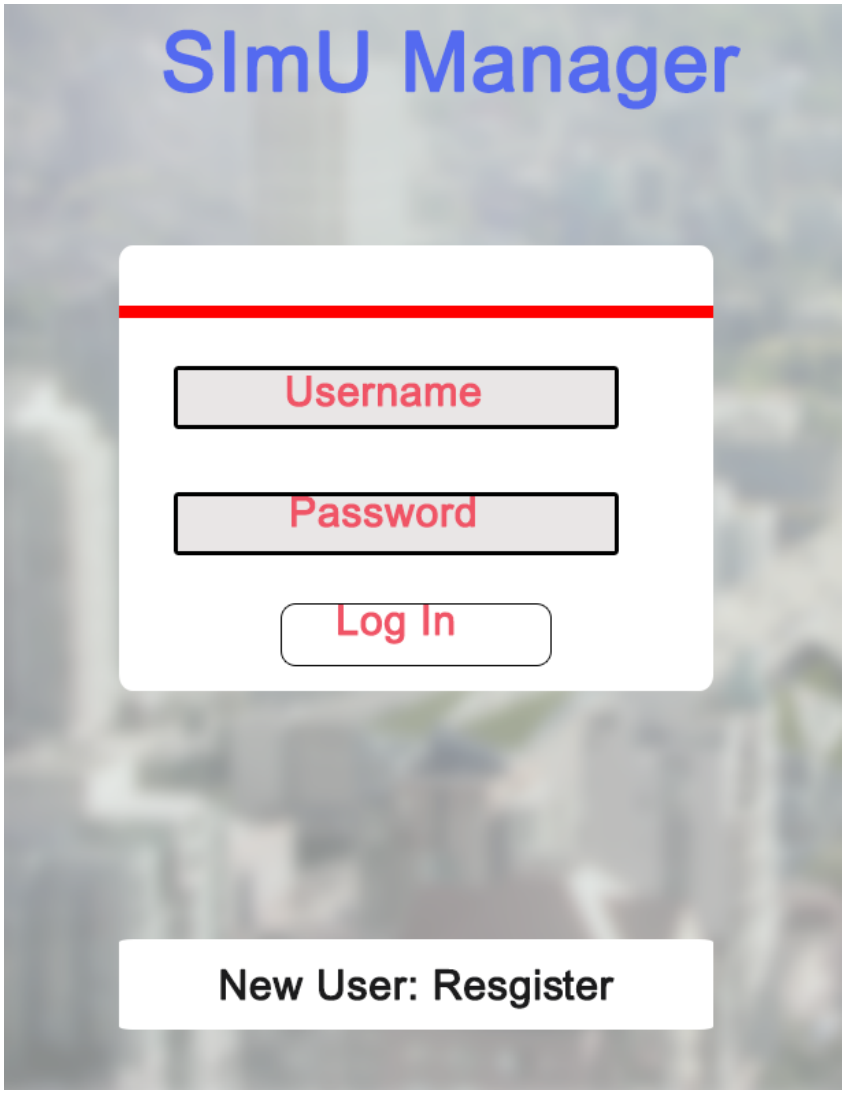
The user device view outlines what a device within a simulation would see. The top bar states the simulation in which the device is in, as well as the name of the device. At the bottom of the page, "network logs" indicates all networks which this device is connected to. The "App: Photoviewer" is an example of an HTML5 application which the admin may have supplied to the simulation. This app includes a local and global count which states the number of times a button has been pressed locally and globally respectively. The left bar indicates various activities that the user may perform.

Register a device to a simulation

Register
Simulation 002



This sketch details the case when a user is allowed to input his or her email in order to receive a token. Once they have received an email they may input the unique code it contains into the “enter token here” box and receive a unique token attached to that browser on that device.



This sketch shows the login page for administrators to the simulation. An administrator will have a unique username and password they will use in order to login or log out and must register in order to obtain it.

This sketch outlines a user attempting to register for a simulation. It allows for a user to view the possible simulations that the devices may register to and allows a user to propagate an HTML5 application. Once a “register” button has been pressed on a simulation, it will allow a user to enter a token that they have been given in order to join said simulation.

RESTful JSON API

We include the following overview of our application programming interface in order to explain the calls and interfaces that we are employing in our program. This is aimed to give a comprehensive overview of all calls and explain their meanings.

Authenticate

Method - authToken
URL: /authenticate/authToken
Parameters: token
Request Type: GET
Format: {‘token’: “token unique ID”}
Response: { ‘Response’: ‘Success’}
Response: {‘Response’: ‘Fail’}

This call calls the server to verify the validity of a supplied token from the client. This will be used both in the situation of when a client enters a unique key to have a token assigned to their device, and the case when a device access a simulation and requires their token to be verified.

Method - authenticate/userLogin
Parameters: username, password
Request Type: GET
Format: {username: ‘ooo524’, password:’noneofyourbusiness’ }
Response: { Response: ‘Success’, client-auth-id: ‘kkj02’}
Response: { Response: ‘Fail’, client-auth-it: ‘’}

This call is to the server to verify the validity of a username and password pair from the client to the server. This is used in order to verify that the device accessing the application is an administrator.

Add

Method - AddDevice2Network
URL : /add/Device/Network
Request Type: POST

Parameters - : device_name, network_name, partition_name, simulation_name, token
Body Format:
{ token: ‘blahblah’, partition_name: ‘Partition-1’, device_name: 45, network_name: 344, simulation_name : ‘Sim02’, config_map:{configmap format}}

This method informs the server that a user would like to add their device to a particular network in a particular partition in a particular simulation. This method requires the user’s token in order to verify validity of the device.

Method - Add2FreeList
URL : /add/Device/FreeList
Request Type: POST
Parameters - : device_name, simulation_name, token
Body Format:
{ token: ‘blahblah’, device_name: 45, simulation_name : ‘Sim02’}

This method informs the server that a user would like to add their device to the free list partition. This is a partition which has no networks and exists outside of all other networks and partitions in the simulation. This method requires the user’s token in order to verify validity of the device.

Create

Method - createNetwork
URL: /create/Network
Request Type: POST
Parameters: network_name, simulation_name, partition_name, token
Body Format:
{ network_name: ‘344’, simulation_name: ‘Sim02’, partition_name: ‘Parta’ token: ‘tokenID’}

This call informs the server that a new virtual network should be created within a partition. In the case that it is a device which is creating this virtual network, the token of that device is supplied in order to allow that device to manage that network.

Method - createDevice
URL: /create/Device
Request Type: POST
Parameters: device_name, simulation_name
Body Format:
{ device_name: ‘devicew@mun.ca’, simulation_name: ‘Sim02’}

This call informs the server that a new virtual device should be created within a partition.

Method - CreateSimulation
URL: /create/Simulation
Request Type: POST
Parameters: name, email_list, num_devices, num_network, config_map, tokenMethod
Body Format:

```
{ name: 'Sim02',
 num_devices: 11,
num_networks: 5,
tokenMethod: 'email'
config_map: {
        'Partition1':
                {'networka' :
                        { 'devicea' : 1,  'deviceb@mun.ca': 2, 'devicec@mun.ca':3},
                'networkb' :
                        { 'deviced': 4, 'devicee': 5},
                                },
        'Partition2':
                { 'networkc' :
                        { 'devicef': 6, 'deviceg@mun.ca' : 7,  'deviceh@mun.ca': 8},
                'networkd' :
                        {'devicei@mun.ca':9, 'device@mun.ca': 10},
                                },
        'Partition3':
                { 'networke' : { 'devicek':'11'} }
       freelist : {devicew: 13, devicex : 14}
}
```

This call informs the server that a new simulation is to be created. It includes the partitions, networks, and devices which should be part of this simulation. For each device, an email is entered, which will be used to propagate the token to.

Method: mergePartitions:
URL: /merge/Partitions
Parameters: partition_a, partition_b, simulation_name
Reqest Type: POST
Format:

```
{ partition_a: 'parta',
partition_b : 'partb',
        simulation_name : 'Sim01',
}
```

This call informs the server that a new partition from two partitions is to be created. The partition is created by merging the two partitions together

Delete

Method - removeDevice
URL : /remove/Device/Network
Request Type: POST
Parameters - : device_name, network_name, partition_name, simulation_name
Body Format:

{ partition_name: ‘Partition-1’, device_name: 45, network_name 344, simulation_name : ‘Sim02’}

This method informs the server that a user would like to add their remove a device from a network to a particular network in a particular partition in a particular simulation. This method requires the user’s token in order to verify validity of the device.

Method - removeDevicefromFreeList
URL : /remove/Device/FreeList
Request Type: POST
Parameters - : device_name, simulation_name, token
Body Format:

{ token: ‘blahblah’, device_name: 45, simulation_name : ‘Sim02’}

This method informs the server that a user would like to remove their device to the free list partition. This is a partition which has no networks and exists outside of all other networks and partitions in the simulation. This method requires the user’s token in order to verify validity of the device.

Method - deleteDevice
URL: /delete/Device
Parameters: device_name, simulation_name
Request Type DELETE
Body Format:

{ device_name: ‘45’, simulation_name : ‘Sim02’ }

This call informs the server that a device should be deleted from the simulation.

Method - deleteNetwork
URL: /delete/Network
Parameters**:** network_name, simulation_id

Request Type: DELETE
Body Format:
{ networkname ‘344’, simulation_name : ‘Sim02’ }

This call informs the server that a networks should be deleted from the simulation.

Method - deleteToken
URL: /delete/Token
Parameters: token
Request Type: DELETE
Body Format:
{token: ‘blahblah’ }
This call informs the server that a certain token should be removed from the database. All information with respect to this token is then wiped, including the device connected to it.

Method: deletePartition:
URL: /delete/Paritition
Parameters: partition_name, simulation_name, token
Reqest Type: DELETE
Format:
{ partition_name: ‘parition02’,
simulation_name : ‘Sim02’,
token: ‘blahblah’,
}

This call informs the server that it should delete a particular partition of the server. This will delete all networks within this partition as well.

Method: deleteSimulation
URL: /delete/Simulation
Parameters: simulation_name
Reqest Type: DELETE
Format:
{ simulation_name : ‘Sim02’,}

This call tells the server to delete an entire simulation. This deletes all data connected to this simulation including tokens, networks, partitions, and devices.

Update

Method - updateLocalCount

URL: /update/LocalCount
Parameters: localcounter, token, current_network, simulation_name
Request Type: POST
Body Format:
{token: ‘blahblah’, localcounter: 345, current_network: ‘home’, ‘simulation_name’ : ‘Sim02’ }

This method tells the server that the local count with respect to this device has been updated for this user with this specified token.

Method - updateNetworkName
URL: /update/NetworkName
Parameters: old_name, new_name, token, simulation_name
Request Type: POST
Body Format:
{token: ‘blahblah’, old_name: ‘oldie’, new_name: ‘newbie’, simulation_name: ‘Sim02’ }

This call tells the server to update a particular name of a network with a new supplied network name.

Method: updateDeviceName
URL: /update/DeviceName
Parameters: old_name, new_name, token, simulation_name
Request Type: POST
Body Format:
{token: ‘blahblah’, old_name: ‘oldie’, new_name: ‘newbie’, simulation_name: ‘Sim02’ }

This call tells the server to update the name of a particular device with the supplied name.

Method: updateSimulationName
URL: /update/SimulationName
Parameters: old_name, new_name, simulation_name
Request Type: POST
Body Format:
{ old_name: ‘oldie’, new_name: ‘newbie’, simulation_name: ‘Sim02’ }

This call tells the server to update this particular simulations name with the supplied name.

Method: updateTokenMethod
URL: /update/TokenMethod
Parameters: new_method, simulation_name
Request Type: POST
Body Format:

{ simulation_name: ‘Sim02’, new_method: ‘newbie’ }

This call tells the server to update the method by which tokens are sent to users with the supplied method.

Method: updateDeviceNumber:
URL: /update/DeviceNumber
Parameters: device_number, simulation_name
Request Type: POST
Body Format:
{ device_number: 12, simulation_name:’Sim02’ }

This call tells the server to update the number of devices in a simulation

Method: updateNetworkNumber
URL: /update/NetworkNumber
Parameters: network_number, simulation_name
Request Type: POST
Body Format:
{ network_number: 12, simulation_name: ‘Sim01’ }

This call tells the server to update the number of networks in a particular simulation.

Method: updateConfigMap:
URL: /update/ConfigMap
Parameters: partition_name, config_map, simulation_name
Reqest Type: POST
Body Format:

```
{ PartitionName: ‘parition02’,
        simulation_name : ‘Sim01’,
                ,config_map:
                        { ‘networka’ :
                                { ‘devicef’: 1, ‘deviceg’: 2, ‘deviceh’: 3},
                        ‘networkd’ :
                                {‘devicei’: 4, ‘devicej’: 5},
                        },
}
```

This call tells the server to update a particular partition configuration within a particular simulation.

Method: divide/Partition:

URL: /divide/Partition
Parameters: partition_name, network_name , simulation_name
Reqest Type: POST
Format:

```
{ partition_name: 'networka',
        simulation_name : 'Sim01',
        network: 'networka'
}
```

This call informs the server that a new partition is to be created from a network within the partition

GET

Method: getSync
URL: /get/Sync
Parameters: eventQueue* , token
Request Type: GET
Format:

```
{token : 'blahblah', simulation: 'Sim02' , eventQueue:
        [
                { 'URL' : '/add/Device', 'Body' : format_body, timestamp : '2015-01-09
                12:00:00'},
                { 'URL' : '/add/Device' , 'Body' : format_body,  timestamp : '2015-01-09
12:03:00'},
                {'URL' : '/delete/Device' , 'Body' : format_body,  timestamp : '2015-01-09
        12:23:00'},
                {'URL' : ' /add/Simulation' , 'Body' :  format_body,   timestamp :
        '2015-01-09 12:33:00'},
                {'URL' : '/update/LocalCount', 'Body' :  format_body, timestamp :
        '2015-01-012:44:00'}
        ]
}
```

This method is the core of our system. It sends the status on the client side, including all events which have occurred on the client side since the last getSync to the server. This information is then processed by the server and the server is updated. Once this is done, the current global status of the simulation is returned to the device so update the device.

Method: getSimulation
URL: /get/Simulation

Parameters: simulation_name
Request Type: GET
Format:
{client_id : 'blahirinr902enrg', simulation_name : 'Sim02'}

Method: getStates
URL: /get/States
Parameters: client_id, simulation_name
Request Type: GET
Format:
{client_id : 'blahirinr902enrg', simulation_name : 'Sim02'}

***see above for the format body of the event queues**

RESPONSE : The Application state is returned as a JSON object for all calls

links: http://crunchify.com/how-to-iterate-through-jsonarray-in-javascript/
http://code.flickr.net/2008/09/26/flickr-engineers-do-it-offline/
http://code.flickr.net/2009/03/18/building-fast-client-side-searches/

Example 1: A typical user. The token is used to compare the values in the database and gives the user a state based on this and the system and simulation state. Don't worry about the clientid field, as no one will have one until we use start getting access rights such as administrators, super admins, etc.

```
{  appstate : {
        device : {
            token:  'blahblah'
            email: 'ooo524@mun.ca',
            verified : true,
            current_partition: 'Partition_1',
            current_network : 'networka',
            current_simulation: 'Sim02',
            registeredOn : '2015-01-09',
            is_admin : false,
            networks_created: ['networke'],
            application_id: 'default',
            localcount: 23,
            globalcount: 345,
            current_device_name: 'device11',
```

```
        activity: '',
},
current_simulation_session : {
        num_devices: 11,
        id : 'blahdu3'
        num_networks: 5,
        simulation_name: 'Sim02',
        config_map: {
        'Partition1':
                {'networka' :
                        { 'devicea' : 1,  'deviceb@mun.ca': 2, 'devicec@mun.ca':3},
                'networkb' :
                        { 'deviced': 4, 'devicee': 5},
                },
         'Partition2':
                { 'networkc' :
                        { 'devicef': 6, 'deviceg@mun.ca' : 7,  'deviceh@mun.ca': 8},
                'networkd' :
                        {'devicei@mun.ca':9, 'device@mun.ca': 10},
                },
         'Partition3':
                { 'networke' : { 'devicek':'11'}
        }

        freelist : { 'devicef': 11, 'devicen@mun.ca' : 12,  'deviceo@mun.ca': 13 }

        },

        globalcount: 230,
        simulation_population: 3,
        token_list {},
        activity_logs: 'blahblah some stuff happened in this simulation, data only about
        your actions '
},
application: {
        simulation_list:  [{name : 'Sim01' , num_devices: 11, num_networks: 5},
                           {name: ''Sim02', num_devices: 45, num_networks: 4},
                            {name: 'Sim03', num_devices: 9,num_networks: 2},
                            {name: 'Sim04', num_devices: 7,num_networks: 2},
                             {name: 'Sim05' ,num_devices: 2, num_networks: 8}]
        super_admin: {},
        total_devices : 68,
        total_networks: 23,
```

```
        },
        'states' :
                {id : 'blahdu3',
                        state:  [{ 'id' : simulation_session, timestamp: '2015-01-012:44:00'
                                'devices': [{ device: device_object},{ device: device_object},{
                                device: device_object},{ device: device_object}],

                                {'id' :  simulation_session, timestamp: '2015-02-013:44:00',
                                'devices': [{ device: device_object},{ device: device_object},{
                                device: device_object},{ device: device_object}],


                                { 'id' : simulation_session, timestamp: '2015-02-022:44:00' ,
                                'devices': [{ device: device_object},{ device: device_object},{
                                device: device_object},{ device: device_object}],
                                } ]
                }
    }
```

Example 1-a: The same as before but the user is only registered in one simulation. The token is used to compare the values in the database and gives the user a state based on this and the system and simulation state.

```
{  appstate : {
        device : {
                tokens:  'blahblah',
                email: 'ooo524@mun.ca',
                verified : true,
                currentpartition: 'Partition_1',
                currentnetwork : 'networka',
                networks_created: [],
                currentsimulation: 'Sim02',
                registeredOn : '2015-01-09',
                is_admin : false,
                application_id: 'default',
                current_device_name: 'device11',
                activity: '',
        },
        current_simulation_session : {
                num_devices: 11,
                id : 'blahdu3',
                num_networks: 5,
                simulation_name: 'Sim02',
```

```
        config_map: {
            {‘Partition_1’,
                { ‘networka’ :
                        { ‘devicea@mun.ca’ : ‘1’,  ‘deviceb@mun.ca’: 2,
                    ‘devicec@mun.ca’:3},
                ‘networkb’ :
                        { ‘deviced@mun.ca’: 4, ‘devicee@mun.ca’: 5},
                },
            Partition_2:
                { ‘networkc’ :
                        { ‘devicef@mun.ca’: 6, ‘deviceg@mun.ca’ : 7,
                    ‘deviceh@mun.ca’: 8},
                ‘networkd’ :
                        {‘devicei@mun.ca’: 9, ‘devicej@mun.ca’: 10},
                },
             Partition_3:
                { ‘networke’ :
                        { ‘devicek@mun.ca’:11}
                }
            freelist : { 'devicef': 11, 'devicen@mun.ca' : 12,  'deviceo@mun.ca': 13 }

        ]},
        globalcount: 230,
        simulation_population: 3,
        token_list {},
        activity_logs: ‘blahblah some stuff happened in this simulation, data only about
        your actions ‘
    },
    application: {
        simulation_list:  [{name : ‘Sim01’ , num_devices: 11, num_networks: 5},
                    {name: ‘‘Sim02’, num_devices: 45, num_networks: 4},
                      {name: ‘Sim03’, num_devices: 9,num_networks: 2},
                      {name: ‘Sim04’, num_devices: 7,num_networks: 2},
                       {name: ‘Sim05’ ,num_devices: 2, num_networks: 8}]
        super_admin: {},
        total_devices : 68,
        total_networks: 23,

    },
    ‘states’ :
        {id : ‘blahdu3’,
`           state:  [{ ‘id’ : simulation_session, timestamp: ‘2015-01-012:44:00’
```

‘devices’: [{ device: device_object},{ device: device_object},{ device: device_object},{ device: device_object}]

{‘id’ : simulation_session, timestamp: ‘2015-02-013:44:00’, ‘devices’: [{ device: device_object},{ device: device_object},{ device: device_object},{ device: device_object}]

{ ‘id’ : simulation_session, timestamp: ‘2015-02-022:44:00’ , ‘devices’: [{ device: device_object},{ device: device_object},{ device: device_object},{ device: device_object}]
}]

}

}

Example 2: A new user. The Token is used to compare the values in the database and gives the user a state based on this and the system and simulation state.

```
{  appstate : {
        device : {
            tokens: ‘’,
            email: ‘’
            verified : false,
            currentnetwork : ‘’,
            currentpartition: ‘’,
            currentsimulation: ‘’,
            registeredOn : ‘2015_01-09’,
            networks_created: [],
            is_admin : false,
            application_id: ‘default’,
            current_device_name: ‘’,
            activity: ‘’
        },
        current_simulation_session : {
            num_devices: 0,
            num_networks: 0,
            id : ‘blahdu3’,
            simulation_name: ‘’ ,
            config_map: {},
```

```
            globalcount: 0,
            simulation_population: 0,
            token_list {},
            activity_logs: ‘’
        },
        application: {
            simulation_list:  [{name : ‘Sim01’ , num_devices: 11, num_networks: 5},
                               {name: ‘‘Sim02’, num_devices: 45, num_networks: 4},
                                {name: ‘Sim03’, num_devices: 9,num_networks: 2},
                                {name: ‘Sim04’, num_devices: 7,num_networks: 2},
                                 {name: ‘Sim05’ ,num_devices: 2, num_networks: 8}]
            super_admin: {},
            total_devices : 68,
            total_networks: 23,

        },
        ‘states’ :
            {id : ‘blahdu3’,
`                       state:  [{ ‘simulation’ : simulation_session, timestamp:
‘2015-01-012:44:00’
                                ‘devices’: [{ device: device_object},{ device: device_object},{
                                device: device_object},{ device: device_object}]

                                {‘simulation’ :  simulation_session, timestamp:
                            ‘2015-02-013:44:00’,
                                ‘devices’: [{ device: device_object},{ device: device_object},{
                                device: device_object},{ device: device_object}]


                                { simulation’ : simulation_session, timestamp:
                            ‘2015-02-022:44:00’ ,
                                ‘devices’: [{ device: device_object},{ device: device_object},{
                                device: device_object},{ device: device_object}]
                                } ]

        }


}
```

Example 3: A super user. The user’s password and username is used to compare the values in the database and gives the user a state based on this and the system and

simulation state. The username and password is swapped for a client side clientid to identify the user.

***** Should not be implemented as of yet ******


```
{  appstate : {
        device : {
                tokens: ‘’,
                email: ‘’
                verified : true,
                currentnetwork : ‘’,
                simulation: ‘’,
                currentpartition: ‘’,
                networks_created: [ ],
                registeredOn : ‘2015-01-09’
                is_admin : true,
                application_id: ‘default’
                current_device_name: ‘’,
                activity: ‘’,
        },
        current_simulation_session : {
                num_devices: ,
                num_networks: ,
                simulation_name: ‘’,
                id : ‘blahdu3’,
                config_map: {},
                globalcount: 0,
                simulation_population: 0,
                token_list { },
        },
        application: {
                simulation_list:  [{name : ‘Sim01’ , num_devices: 11, num_networks: 5},
                                  {name: ‘‘Sim02’, num_devices: 45, num_networks: 4},
                                   {name: ‘Sim03’, num_devices: 9,num_networks: 2},
                                   {name: ‘Sim04’, num_devices: 7,num_networks: 2},
                                    {name: ‘Sim05’ ,{num_devices: 2, num_networks: 8}]
                super_admin: {},
                total_devices : 68,
                total_networks: 23
        },
        ‘state’ :
                {id : ‘blahdu3’,
  `                     states:  [{ ‘’ : simulation_session, timestamp: ‘2015-01-012:44:00’
                                ‘devices’: [{ device: device_object},{ device: device_object},{
                                device: device_object},{ device: device_object}]
```

{‘id’ : simulation_session, timestamp: ‘2015-02-013:44:00’,
‘devices’: [{ device: device_object},{ device: device_object},{
device: device_object},{ device: device_object}]

{ ‘id’ : simulation_session, timestamp: ‘2015-02-022:44:00’ ,
‘devices’: [{ device: device_object},{ device: device_object},{
device: device_object},{ device: device_object}]
}]

}

Explanations

Right now when the application starts fresh, the user navigates to the webpage for the first time and the client makes a call at start. This call is to getSync with the server. At this stage the client will not have a token. The will check the token ID sent. If it is empty it sends back this state every time. Later on when we implement client ID, this may change.

```
{  appstate : {
        device : {
            tokens: ‘’,
            email: ‘’
            verified : false,
            currentnetwork : ‘’,
            currentpartition: ‘’
            currentsimulation: ‘’,
            registeredOn : ‘2015-01-09’
            networks_created: [ ]
            is_admin : false,
            application_id: ‘default’
            current_device_name: ‘’,
        },
        current_simulation_session : {
            num_devices: 0,
            num_networks: 0,
            simulation_name: ‘’ ,
```

```
            id : 'blahdu3',
            config_map: {
                map: { }
            },
            globalcount: 0,
            simulation_population: 0,
            token_list {},
            activity_logs: ''
        },
        application: {
            simulation_list: [{ name : 'Sim01' ,
                                num_devices: 11,
                                num_networks: 5}
                                ]
            num_devices : 68,
            num_networks: 23,
            superadmin: {} }
        }
}
```

Create Simulations

If there are no Simulations in the environment that simulation_list at the end will also be blank. Now after the user sends a call to create a simulation. The follow occurs

Step 1:

Initiator: Client

Client sent the request with an event queue. Embedded in that queue is the the URL for /create/Simulation.

Step 2:

Initiator: Server

Server routes the clients request for getSync and parses the token and the eventqueue. Once the server checks the URL contained in the eventQueue it extracts the information as dictated in the above API for how the format_body will look like. The server extracts the format body. This will be of the form:

```
{ name: 'Sim02',
 num_devices: 11,
num_networks: 5,
id : 'blahdu3',
tokenMethod: 'email'
config_map: { Partition_1:
                { 'networka' :
```

{ ‘devicea@mun.ca’ : 1, ‘deviceb@mun.ca’: 2,
‘devicec@mun.ca’:3},
‘networkb’ :
{ ‘deviced@mun.ca’: 4, ‘devicee@mun.ca’: 5},
},
Partition_2:
{ ‘networkc’ :
{ ‘devicef@mun.ca’: 6, ‘deviceg@mun.ca’ : 7,
‘deviceh@mun.ca’: 8},
‘networkd’ :
{‘devicei@mun.ca’: 9, ‘device@mun.caj’: 10},
},
Partition_3:
{ ‘networke’ :
{ ‘devicek@mun.ca’:11}
}
}

The server extracts the necessary fields of email_list, config_map, tokenMethod, num_networks, num_devices, name.
The server then calls the appropriate method indicated by the URL. In this case it is createSimulation(email_list, config_map, tokenMethod, num_networks, num_devices, name).

Step 3:

Initiator: Simulation Manager

Once the Simulation Manager receives the call for createSimulation(params) it needs to do several things:

First, connect to the database and enter the information provided as an object. However it needs to put it back in the database in the format of:

{ name: ‘Sim02’,
num_devices: 11,
num_networks: 5,
tokenMethod: ‘email’
config_map: { Partition_1:
{ ‘networka’ :
{ ‘devicea@mun.ca’ : ‘1’, ‘deviceb@mun.ca’: 2,
‘devicec@mun.ca’:3},
‘networkb’ :
{ ‘deviced@mun.ca’: 4, ‘devicee@mun.ca’: 5},
},
Partition_2:
{ ‘networkc’ :
{ ‘devicef@mun.ca’: 6, ‘deviceg@mun.ca’ : 7,
‘deviceh@mun.ca’: 8},

‘networkd’ :
{‘devicei@mun.ca’: 9, ‘devicej@mun.ca’: 10},
},
Partition_3:
{ ‘networke’ :
{ ‘devicek@mun.ca’:11}
}
free_partition:
}

This is inserted in the collection for simulations. Once this is done, the manager gets the devices in the simulation using an iterator. This iterator will be shared code on the client and server.
For each device it needs to create a new object in the format:

```
user : {
    token: ‘’,
    email: ‘’
    clientid: ‘’, ** don’t worry about how or when we will implement this **
    verified : false,
    currentnetwork : ‘’,
    currentpartition: ‘’
    currentsimulation: ‘’,
    registeredOn : ‘’
    networks_created: {}
    is_admin : false,
    application_id: ‘default’
    current_device_name: ‘’,
},
```

To create this object this is the same as:

var user_data = {}

As shown above verified is obviously false because the user has not registered with the simulation yet. But the device technically exists now in the virtual simulation.
It also needs to modify the following fields in this Object.
name is the name of the simulation provided by the parameters
To set the currentSimulation this user/device has been created in.

user.currentsimulation = name;

To set the currentNetwork _ or default network at start - this user/device has been created in. The client has a function called getparent(node) to access the network name from a given device name. the device name is passed in as the node. This is because config_map is in a tree structure.

user.currentnetwork = getparent(device_name);

We set the current device name as
user.current_device_name = device_name

we set the current partition for the device as
user.currentpartition = getparent(user.currentnetwork)

We set the email of the device to the name of the device as well
user.email = device_name

And Finally most importantly we assign the user the token we just generated. We don't need to actually pass in the simulation name but just as an example. Token generation is a blackbox.
user.token = tokenManager.generateToken(name)

Step 4:
Initiator: Database Manager
Thats it now we ship this to the database.
First wrap it up as a JSON object.
so
for_db_user = JSON.parse(user);
mongo.user.insert(for_db_user);
Do this for each user to be created

Step 5:
Initiator: Simulation Manager
The application must now edited to include the new simulation. A new object must be entered in the array of simulations at this level.
item = { 'name' : *simulation.name*, 'num_devices' : *simulation.num_devices*, 'num_networks' : *simulation.num_networks* }

Step 6:
Initiator: Database Manager
Retrieve the application collection object from the database. This object looks like:

application: {
simulation_list: [{name : 'Sim01' , num_devices: 11, num_networks: 5},
{name: ''Sim02', num_devices: 45, num_networks: 4},
{name: 'Sim03', num_devices: 9,num_networks: 2},
{name: 'Sim04', num_devices: 7,num_networks: 2},

{name: ‘Sim05’ ,{num_devices: 2, num_networks: 8}]
super_admin: {},
total_devices : 68,
total_networks: 23,
}
Then send back the response to the simulation manager.

Step 7:
Initiator: Simulation Manager
Add the item from above in the simulation_list array.
simulation_list.push(item)
edit the total_devices and add the simulation.num_devices to it
application.total_devices += simulation.num_devices
Do the same for the num_networks
application.total_networks += simulation.num_networks

Check above for response format.

Auth Token

Step 1:
Initiator: Client
Client sends the request for authentication called /authenicate/Token with its necessary body: the token.

Step 2:
Initiator: Server
Server parses request, extracts the token from the JSON object and calls the function Simulation.Manager.authenticate(token). The actual function name is arbitrary. What matters is what the function call DOES.

Step 3:
Initiator: Simulation Manager
Simulation manager asks the database manager for the item in the ‘users’ collection that has the token matching the one it has.
If the database returns an empty object then the server must return false, otherwise the server returns true.

Step 4:
Initiator: Server

The server sends this response back as a JSON object

Steps for activity Logs

The application should be able to display the activity logs of the simulation not only within the device view but an overview for the administrator of the simulation.
To implement this we have included the activity parameter for the activity in each device collection to display activities performed by that device. The activity log is also kept for each simulation in the current_simulation collection.

Planning for Iteration 3

The following is a list of tasks we aim to complete for iteration 3. We believe that these are the crucial to the goal of the next iteration as we understand it:

- Clean up main.js. It can be separated into several files according to the modules section, and some functions could use better names.
- Clean up unnecessary javascript source files. We have a lot of files which we had planned to use but we decided that we no longer needed them. They should be removed from the directory.
- Create code to automatically sync the client with the server. Currently the client has to press a button to sync, but this is not ideal.
- Refactor our code to be compatible with the supplied RDTs and the given interface for it.
- Improve network creation interface, allow device users to create their own sub-network. Allow for the deletion of networks. This has been coded for on the server side but has not yet been implemented on the client side.
- Prepare to handle the user created HTML5 applications.
- Refactor CSS to better handle window resizing for varying screen sizes.
- Modify and further refactor server-side code to allow for improved functionality and performance.
- Improve upon the current topology GUI to better suit the wants of the client.