Database

Software : The database was created with mongoose ODM. We chose mongoose for its straightforward, schema based solution to model our application data.

Each major object of our application was designed with mongoose in mind. The application creates objects based from our schema, edits variables of said schema, can use static functions defined in the overall model, and can be saved to our db.

The models are registered and ready for use on start up of a node session. The standard attributes of mongoose are mapped as follows:

NameOfProperty : <Type>

Mongoose has the following defined types:

- String
- Number
- Date
- Buffer
- Boolean
- Mixed
- ObjectId
- Array

When an object is created based off of our designed schemas, an unique object id, declared as _id, is created and is assigned to the object. We use this unique id inside models to reference our objects that are “linked” together.

Connection to Database
Path : /Database/mongooseConnect

On startup our node server creates a constant connection with our database being hosted by mongolab. If mongoose is required, and an object is constructed with a mongoose function the database is listening for input.

Each Model constructor has three variables. A name, a schema reference, and a collections name.

Models - /Database/dbModels

Application Model - */Database/dbModels/appModel.js*

This model is responsible for defining an application object. It has general properties such as name, version, and description of an application. It also includes the application code and a list of RDTS's used.

Application Model constructor

Name: App
Schema: applicationSchema
Collection Name: Apps

ie. mongoose.model('App', applicationSchema, 'Apps')

applicationSchema Properties/* Brief Description

name : String, *Name of an application

version : String, *Version of application

description : String, *Description of application

main : String, *Main of application

rdt_list : [String] *List of rdt's being used by this application

applicationSchema Functions

1. getAppByID(_id, callback)

Function explanations

1. Uses mongoose built in query to search the application collection, “apps”, for a matching object id. If found it returns that object otherwise it prints that no object exists.

User Model - Database/dbModels/userModel.js

This model is responsible for representing our user objects ie. Devices that are using our master application. It has general properties such as user information, user permissions, and keeps record of recent activity from a user.

User Model Constructor

Name : User
Schema : userSchema
Collection: Users

ie. mongoose.model('User', userSchema, 'Users")

userSchema

token:String,	*Server generated user identifier
email:String,	*Email Address of User
verified: Boolean,	*true when user is verified
current_partition: String,	*The current partition of the user
current_network: String,	*The current network of the user
registeredOn: String,	*Date when the user was registered
admin: Boolean,	*True if user is an admin
networks_created: [String],	*List of network created by user

current_simulation: String, *Current simulation of user

current_device_name: String, *Name of user's current device

activity : String, *Record of user activity

apps : [{type : mongoose.Schema.Types.ObjectId, ref: 'App'}]

* This object can reference none to many application objects by object id depending on how the user is using our master application.

userSchema functions

1. getUserByID - Queries the user collection based on objectID and returns a user object if successful.

Network Model - /Database/dbModels/networkModel.js

This model is responsible for defining our network objects. It has general properties such as name and type. Network also has reference to an array of Users object ID. This array represents the users currently with devices' joined to a network.

Network Model constructor

Name: Network
Schema: networkSchema
Collection Name: networks

ie. mongoose.model('Network', networkSchema, 'Networks')

networkSchema Properties/*Brief Description

network_name : String, *name of network

network_type : String, *type of network (wifi or gsm)

device_list : [{type : mongoose.Schema.Types.ObjectId, ref: 'User'}],
*List of devices currently using this network referencing them by ObjectId.

Network Functions

1. getNetworkByID - Queries the Networks collection based on objectID and returns a Network object if successful.

Partition Model - /Database/dbModels/partitionModel.js

This model represents our partition objects. Our partition object represents a transitive relation between a single simulation object and none to many networks. It contains a list referencing network objects.

Partition Model constructor

Name: Partition
Schema: partitionSchema
Collection: partitions

ie. mongoose.model(‘Partition’, partitionSchema, ‘Partitions’)

partitionSchema / *Description

network_list : [{type : mongoose.Schema.Types.ObjectId, ref: 'Network'}],
*A list that references network objects

Partition Functions

1. getPartitionByID - Queries the Partitions collection based on objectID and returns a Partition object if successful.

RDT Model - /Database/dbModels/networkModel.js

This model is responsible for defining our RDT objects. It has general features of name, version, description.

RDT Model Constructor

Name: RDTModel
Schema: RDTSchema
Collection: RDTs
ie. mongoose.model('RDT', RDTSchema, 'RDTs')

RDTSchema

name : String,	*Name of the RDT
version : String,	*Version number of the RDT
description : String,	*Short description of what the RDT does
main : String	*File name/reference

RDT Functions

1. getRDTByID - Queries the RDTscollection based on objectID and returns a RDTobject if successful.

State Model - /Database/dbModels/paritionModel.js

This model is responsible for representing a state of the application. It contains a simulation id, a simulation object, and a timestamp of a simulation.

State Model Constructor

Name: State
Schema: stateSchema
Collection Name: states

ie. mongoose.model('State', stateSchema, 'States')

stateSchema Properties/*Brief Description

simulation_id : String, *A simulation id.

state : [{ simulation: Object, timestamp : String }],
*Simulation object plus a timestamp.

State Functions

1. getStateByID - Queries the States collection based on objectID and returns a State object if successful.
2. findAllStates - Queries the States collection and returns every state stored in the database.

Simulation Model -Database/dbModes/simulationModel.js

This model defines our simulation object. It includes numbers referencing how many devices, aka “Users”, networks, population it is controlling. It also has general properties like name and token method. Simulation Objects reference three different types of stored objects: RDTs, apps, and partitions.

simulationSchema

num_devices: Number, *Number of devices inside a simulation

num_networks: Number, *Number of networks inside a simulation

simulation_population: * Population of the simulation

simulation_name: String, *Set name of Simulation

tokenMethod : String, *How tokens will be propagated

partition_list: [{type : mongoose.Schema.Types.ObjectId, ref: 'Partition'}],
*Reference to a list of partition objects by ObjectID.

apps : [{type : mongoose.Schema.Types.ObjectId, ref: 'App'}], *Reference to a list of app objects by ObjectID.

rdts : [{type : mongoose.Schema.Types.ObjectId, ref: 'RDT'}], *Reference to a list of rdt objects by ObjectID.

activity_logs : String *A parsable string of activity of a simulation

Simulation Functions

1. findAllSimulations - Queries the Simulations collection and returns every, if any, simulation objects.

Our Models form a hierarchy of responsibility with objects only able to reference other objects if necessary for organization and protection.