**Webmachine**

**Developer Notes**

**Refactoring 0.7.0**

**Introduction**

The webmachine platform delivers administrative functions for the Strategic Machines suite of services. This includes functions like

* Custom static site generation for marketing, news and announcements
* Customer registration and network activation
* Member registration
* Product (agent) selection, configuration and subscriptions
* Custom development of virtual agents through the Strategic Machines developer network
* Analytics
* Multi-channel support (future)
* Contact Center integration and deployment (future)

The Strategic Machines vision is to help clients serve their customers effectively through intelligent real-time engagement technologies

**Functional Change Introduced with .7**

The .7 webmachine platform introduces a data model supporting multi-tenancy. This is a requirement of SM services platform for real time messaging and required as well by all administrative processes. In essence:

* The platform has two modes of operation (test and production) as highlighted in platform.json
* The platform points to the physical database which holds the collection of registered customers
* Each registered customer can specify the api for their physical database. Their database instance holds the collections that are unique for their operation on the SM platform
* A customer (network owner) has members, interactions (text messages) and agents.
* A customer network may operate in private or guest mode. Private mode means that anyone texting the network (using the network phone number, web widget, slack channel etc.) would only see a response if they are a registered member of that network. Public mode means the network will handle guests. A public network may have agents that only operate in ‘private’ mode – meaning for registered members only.
* Every registered user of a customer network is a ‘member’ of the network. We can refer to a registered user as a ‘basic member’. There are also multiple types of members, where extended definition of the member is required. Today, the platform recognizes a type of ‘developer’ who is a member of the network but also doing work in the market. With the extended definition of member, we collect additional information on the developer so that they can be assigned work, and paid for work completed
* Each agent is powered by its own unique ai engine classifier, which requires configurations unique to the customer
* Conceptually, each customer could also have their own set of workitems (requisitions for custom agents, for example), pointing to their own private repos which hold their protected code base. In addition, the customer could have their own developer network (this might be a requirement of developers working on their code require specific, additional security checks, or are in fact employees of the customer)

At this stage, I anticipate deploying the platform with only “1 customer of the platform’ (Strategic Machines) operating a ‘market’ for developers and available work requisitions. But the multi-tenant model would support anyone operating in the same mode with the right enablement.

We began development on the webmachine with the perspective that single tenant would be simpler to grasp, build and test. We are obviously introducing complexity in the SDLC with this shift to .7 – especially without devops automation to help manage progress and releases – requiring all of us to communicate intensively and to stay coordinated as we drive to an ‘MVP’.

**Schemas, Models, Data and Multi-Tenancy with .7**

An ambitious objective of this platform is too serve many organizations concurrently who wish to establish Branded (private) networks. Through s shared infrastructure, costs can be reduced for all organizations that board. Just as important, innovation can also be triggered among platform customers by leveraging and improving a common collection of virtual agents -- which are at the heart of the platform services. As a result, ‘serverless apps’ which are powered by ai engines are more widely available at a competitive economic price point, and advancements are accelerated through a common collaborative environment for all participating organizations

This vision requires multi-tenancy. The model for multi-tenancy is physical separation of data while running on a common code base. Each subscriber (a paying customer wishing to establish a network that is private or public) has a dedicated database, holding collections of data that relate to and are owned by the subscriber.

Specific model structure:

* Platform.json
  + This is the file that is loaded when the application is deployed to a server
  + This file hold 2 objects.
    - One object holds the database api for the collection of **test** clients (paying subscribers)
    - The second object holds the database api for the **production** clients
  + The application determines which object to use (test or production) based on the value of process.env.isLIve
* Client.js
  + This collection is contained in and held by the database used by the platform. This collection holds the set of objects with information pertinent to subscribers of the platform (address, contact information, assigned sms number (twilio number), assigned web id token, billing information etc.)
  + Each client object also contains the api for the physical mongodb which contains the remainder of collections related to the customers configurations, messages, agents, etc. for operating their network
  + If the application is running in test mode, the platform.json object with a property of isLive=false will point to the physical database holding test clients. If production, the platform.json object with a property of isLIve=true will point to the physical database holding the set of production clients.
* Remainder of collections for a client
  + Config – specific keys and tokens related to services that are unique for that customer, such as github repos, Watson services etc.
  + Agents – the collection of agents configured for the clients interactive messaging network
  + Interactions – The collection of all messages sent and received on the customer’s network
  + Workitems – The collection of workitems posted by the client to the market for developers to build. The customer Strategic Machines is the only platform user as of Q218 to deploy and operate an ‘outcomes delivery market’. This market is public and accessible by anyone registered to the platform
  + Members – profiles of individuals who are registered to a client’s network. An individual may have ‘basic membership status’ (can operate as an active user within the customers network) or may have extended status, like ‘partner’, able to conduct work posted to the market. Other ‘member types’ may be identified over time by customers using the platform (note that the test data associated with partners will eventually be populated through a schema reference with members)

For developers, the logic associated with multi-tenancy is more demanding. The requirements for maintaining a clean set of test data is trickier.

* Notes on seedTestDb.js
  + This function is triggered when the platform is started if the environment is detected as ‘test’ (isLIve=test)
  + All test databases are dropped.
  + The test datasets contained in db/data directory are loaded in the following order
  + The client.js collection is copied to the mongodb api retrieved from the platform.json object with a value of isLive=test
  + For each client.js object with a valid mongodb url, the remaining collections are copied to the db end point for that client
  + Note the complexity of the workitem.js model. It must be coordinated with a set of test Repos
  + When the test data is loaded, each client with a valid db api will have an exact copy of the test data contained in the db/data directory

The current test data load process will be improved with time. The db apis for platform.json and client.js point to a private mlab account. These apis should be swapped out for your own mlab account so that you can access and view the test data

* Notes multitenancy and dynamic db connections
  + In seedTestDb.js, calls are made to a series of functions that are found in the db/initialize directory
  + Note that the connection object that is passed to the functions has been created through mongoose.createConnection(). This connection method in mongoose permits database services to be dynamically reallocated, rather than binding a model to a single connection
  + The ability to dynamically bind a service to a db and model based on the api endpoint of the client is foundational to multitenancy.

**Scenarios (? Indicates that not all requirements fully parsed. Needs discussion)**

* A website visitor browses available information on the site, such as solutions, documentation, announcements and news. No registration needed.
* A website visitor Registers to create a network – which would require going through the platform registration process and creates documents in the client collection
* Separately, a network owner indicates they want to be a member of the network as well? A document would be created with basic information would be created in the customers member collection. Note that cell phones need 2fa
* A network owner logs in to manage information. This might also include managing their member data?
* A network owner logs in to generate a static website, view agent gallery, configure new agents, manage members on the network, view analytics of recent traffic and interactions
* A network member wishes to view a gallery of other members of their network?
* A website visitor wishes to register for a network.
  + Enter cell phone number to view pending network invitations?
  + Enter a network name and proceed to register for the site (a registration may be pended until approved)?
  + Browse public networks which they can join?
* A website visitor wishes to register for a network and notes that one of the networks is Strategic Machines – the developer network. (this network in some respects demonstrates the power of a messaging network to conduct commerce) Registration by the visitor is pended until approved (one the ‘modes’ for registration). Note that Strategic Machines as a customer of the platform has already registered and created the work through the platform registration process.
* A website visitor who logs in with credentials authenticating them as a member of the strategic machines (sm) network will be granted access to other web site pages, including workitem browsing and selection?
* The owner of the platform is the super user with admin privileges across the platform, including customer registration data and member data of customers? (runs a network of networks?)
* The owner of the strategic machines platform has admin privileges to certain web pages for creating and maintaining work items?
* An authenticated network owner wishes to configure ‘interactions’ for their network. This will include scripting, agent definition and configurations, testing etc. This may require an offline process with custom specs?
* An authenticated network owner wishes to browse the agent gallery and identify potential participants in their network. Deployment of selected agents may (will) require an offline process with custom specs?
* A website visitor can browse the agent gallery and interact. Part of the marketing process? The ‘deploy me’ button takes them to the network registration page
* The platform registration process is handled through slack (the ui) with slack bot agents integrated with the platform handling the collection of information?
* Members to a network register through slack – interacting with platform agents. The lack account is owned by the network owner and can be used for additional communications and posting of information (as we do for our ‘developer team’ on strategic machines)?