ZeroToBlockchain Chapter 5: Building the Admin interface

Return to Table of Contents

In this chapter, we will build a nodeJS web server which will interact with the HyperLedger Composer defined Blockchain and create an administrative interface for the network. This runs on HyperLedger Fabric Version 1.0. The code that we will write is managed in this chapter and all the subsequent chapters in the following folder structure:

```
Chapter 05
  ↳ controller
     ↳restapi
       router.js
        ↳features
           ↳composer
             autoLoad.js
             hlcAdmin.js
             queryBlockChain.js
             Z2B_Services.js
             Z2B_Utilities.js
           ↳ text
               multi-lingual.js
               resources.js
  ↳ HTML
     ↳CSS
       pageStyles.css
     ↳js
       z2b-admin.js
       z2b-events.js
       z2b-initiate.js
       z2b-utilities.js
  ↳ network
```

```
    permissions.acl
    queries.qry
     ↳lib
       sample.js
     ↳models
       base.cto
       events.cto
       sample.cto
```

Some of these files are used by all chapters and some are specific to this chapter. Code explanations are embedded in the javascript files. Those files are:

Common Web Server code:

- **index.js**
  - this is the master, or controller, file for the web server. It loads a file called 'router.js' to implement all business functionality.

creation, which are called by autoLoader, hlcClient, hlcAdmin, queryBlockChain modules

- **multi-lingual.js**
  - This module contains services to support multiple languages on the browser. Language text is provided in subfolders in the composer/text folder. Which languages are available is managed in the languages.json file.

Common Web Browser code:

- **z2b-utilities.js**
  - This file contains routines which are used by the browser javascript.
- **z2b-initiate.js**
  - This file contains the code necessary to start up the web application.
  - This is where the default starting language is specified

Files used in this chapter

Web Server Code Unique to this Chapter

- **hlcAdmin.js**
  - This contains all code to support the administrative interface on the server side

Defining the business network

- **network/model/sample.cto**
  - The CTO file contains the network description and includes member (buyer, seller, provider, shipper, financeCo), asset (Order) and transaction definitions. The sample.cto file uses two other files as it's foundation: **base.cto** and **events.cto**
- **network/lib/sample.js**
  - This contains the transaction code which will be deployed to

the blockchain. Although normally written in GO, hyperledger composer allows us to write all of the blockchain code in javascript and then uses a plug-in called DukTape to enable the javascript to be executed within the GO environment. This file must implement transactions as defined in the sample.cto file and is one of the places where we ensure that members have the authority to execute the requested transaction.

- **network/permissions.acl**
    - This file determines precisely what transactions a member can execute and what assets a member can see. The basic rule is "If authority is not specifically granted in this file, then access is denied."

Web Browser Code

- **z2b-admin.js**
    - This file contains all of the browser functional code to support the administrative interface **Please Note** that with the update to hyperledger-composer v0.15, composer has transitioned from using connection profiles to using idCards. This means that the connection profile creation work is no longer required to connect to the network.
- **index.html**
    - This file is the initial web page loaded by the application.
    - Text on this page is determined by the selected language (default is US English)
- **admin.html**
    - Contains the HTML to manage the admin interface
    - Text on this page is determined by the selected language **Please Note** that with the update to hyperledger-composer v0.15, composer has transitioned from using connection profiles to using idCards. This means that the connection profile creation work is no longer required to connect to the

network. Therefore, these items now show a yellow strike-through in this chapter and will not be in chapters 6-12.

- **ceateConnectionProfile.html**
  - Contains the HTML to manage profile creation
  - Text on this page is determined by the selected language
- **createMember.html**
  - contains the HTML to manage member creation
  - Text on this page is determined by the selected language
- **deleteConnectionProfile.html**
  - contains the HTML to manage connection deletion
  - Text on this page is determined by the selected language
- **getMemberSecret.html**
  - contains the HTML to manage and display member secrets
  - Text on this page is determined by the selected language
- **removeMember.html**
  - contains the HTML to remove a member
  - Text on this page is determined by the selected language

Where are the Answers?

Chapterxx/Documentation/answers

- all answers for each chapter are in the Documentation/answers folder in that chapter.
- The naming conventions used are:
- ***Folder Name:***
  - composer: these files are the answers to files located in the following path within each chapter: controller/restapi/features/composer/
  - cto: network/model
  - CSS: HTML/CSS
  - HTML: /HTML/
  - js: HTML/js/

    - lib: network/lib
    - test: network/test
- ***File Name:***
    - each file has the same prefix in the answers and in the target folder. The answers folder adds '_complete' to the end of the first part of the file name. For example:
    - the server code for administration is in the ***controller/restapi/features/composer*** folder and is called: hlcAdmin.js. The completed code for that file (in this chapter) is in the ***Documentation/answers/composer*** folder and is called hlcAdmin_complete.js

How do I get everything started once I've updated all the code?

You need to run the following command **once** for each chapter: npm install. This installs all of the node modules required to run your program.

Each time you restart your system, restart Docker, or bring your system back from a 'suspend' state, you need to run the following two commands in this order: - buildAndDeploy (OSX) ... or ... ./buildAndDeploy (Ubuntu) - npm start (or npm index)

The buildAndDeploy command creates the business archive, stops and restarts the docker containers and then deploys your network into those containers.

The npm start command loads and starts to execute the index.js file in your chapterxx folder. This, in turn, loads all of the services in the controller folder structure.

Why All The Changes?

HyperLedger Composer has gone through two significant changes in 4Q2017.

- The first, Version 0.14, changed how security worked and clarified the difference between the business network administrator (admin) and the fabric Administrator (PeerAdmin). While this had little effect on what the Admin user interface could do, it did change significantly how the underlying code worked in the hlcAdmin.js file and in the autoLoad.js file. The biggest code change is that the autoLoad process started issuing Identities instead of just adding members to the registry. The other significant change required adding two statements to the permissions.acl file.

- Then Version 0.15 was released and the entire access structure was changed from connection profile + username/password (which is why we had the getSecret() method) to idCards. This necessitated changes in some of the scripts we use (buildAndDeploy, startup.sh, deployNetwork.sh), required a complete rewrite for how the test code loads and connects to the network and also required changes in the admin interface. You'll see a number of items which have a -->yellow strikethrough<--, indicating that they are no longer available.

- We aren't using connection profiles any more as part of the log in process, so that whole section is no longer useful. Because we're using cards, we don't need to use, nor do we need to capture, the user secrets associated with each Member. so that is deactivated.

We do, however, need more capability for Members. Along with the Add Member function, which we've had all along, we now need the ability to issue an Identity for that Member and then to create an idCard for that member. You'll see those new functions in the Resource Management part of the administrative user interface, along with a feature which will check to see if an idCard already exists for a specific member.

An Identity can only be issued for an existing Member, so the process is to:

- Add a Member
- Issue an Identity
- Create an idCard

Creating an idCard requires the password, or secret, which was generated during the issue Identity process. Each of these functions has been implemented individually. It would be a great idea to combine the issue Identity process with the Create idCard process, so that you don't have to remember to copy the generated secret and type it into the createCard page.