

# Blockchain Do It Yourself Manual

---

DECEMBER 2018

---

## TABLE OF CONTENTS

---

<b>Introduction</b>	<b>3</b>
<hr/>	
<b>Build our blockchain demo application</b>	<b>5</b>
1. The use case: Car tires	6
<hr/>	
2. The issues: Complex bookkeeping and fraud	7
<hr/>	
3. Why and how blockchain can be applied	7
<hr/>	
4. Building a blockchain business network with Hyperledger Fabric	10
<hr/>	
<b>Build your own blockchain application</b>	<b>12</b>
1. Create your own blockchain network	14
<hr/>	
2. About support	26
<hr/>	

---

There is no shortage on blockchain news, one of the most hyped technologies of 2018. It is, or was, supposed to "revolutionize the world economy" but so far we haven't seen a mass adoption of this technology. This has led to a lot of criticism on the real potential of blockchain which is, if you're a believer of hype cycles, nothing to worry about. Technologies come, go and reach maturity over time and we see no reason to assume that this is will be any different for blockchain.



What we learned these last 4 years is that actual working applications are a great way of learning what blockchain can do. At first such an experiment would come at a significant cost and would not deliver all the desired learnings. As a response we started to develop a toolkit that would allow us build a simple blockchain-based solution that would offer as much insights as possible at an unprecedented value. We succeeded and thanks to our toolkit we were able to deliver a proof-of-concept in one week for as little as € 9.990.

2019 Will be the year where more applications based on blockchain technology will surface and we're currently working with our customers to make this happen. We want everyone to understand what blockchain has to offer and we feel the best way to do so is to open source our toolkit so you can try blockchain for yourself. We hope our blockchain toolkit will increase your understanding of blockchain technology without taking too much of your time, or breaking your bank.



---

Our toolkit helped many customers and now it is available for you to download for free. This manual explains how you can create a blockchain network and how you can customize it to fit your needs. Each chapter starts with an explanation and provides a list of actions you should perform to use the template. First we explain our recycle business case to you in detail and provide instructions to get it up and running on your own Mac or PC so you can try it out for yourself. If you're interested to build your own blockchain application the rest of the manual introduces you to our sample project on GitHub which you can use as a boilerplate for your own application. Everything is explained step by step and the deployment processes is fully automated which will save you a lot of time.

Please keep in mind that this toolkit is meant to be used in the first phase of blockchain projects. We strongly advise not to use any application that is built with this toolkit in production. The sole purpose of this toolkit is to experiment with blockchain technology and to build applications for demo purposes. By walking through the steps you'll also learn how to use the Hyperledger Composer development tools.



**Build our blockchain  
demo application**

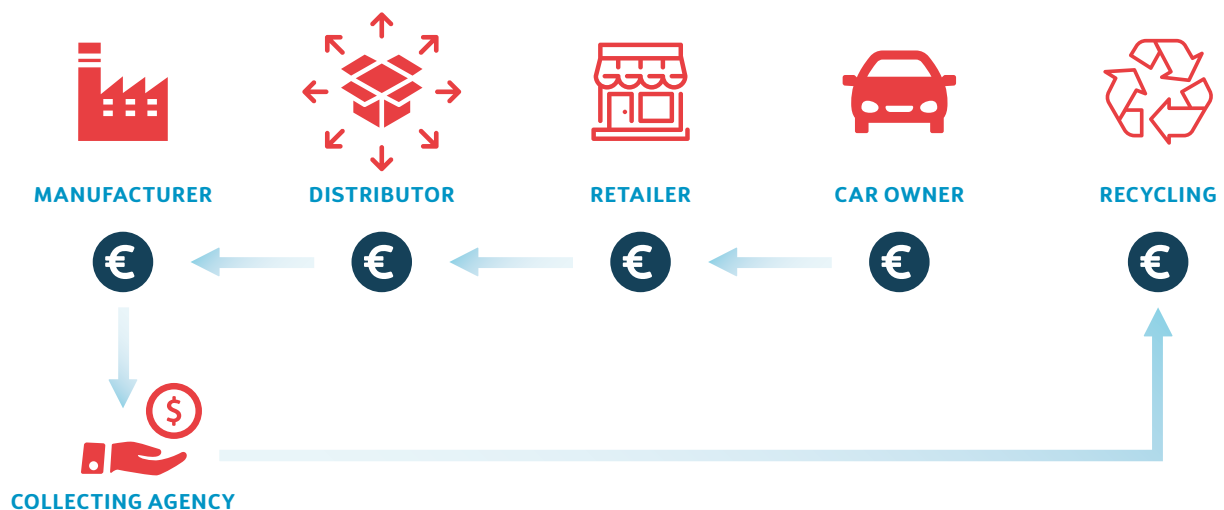


Blockchain technology is perceived by some<sup>1,2</sup> to solve long outstanding issues in the supply chain of waste management. So far the adoption of this technology is not yet widespread but there are some interesting solutions that show great potential such as the one built by Circularise<sup>3</sup>. In this manual we outline how blockchain technology can be used to a) decrease cost by reducing administrative overhead and b) incentivize parties to conduct their business more honestly in the recycling industry. We will translate our ideas into a working application using our toolkit to explain the impact of this new and exciting technology.

## 1. The use case: Car tires

In countries where the collection of used car tires is regulated there is usually a (non-profit) organization that collects fees when car tires are imported and uses these fees to pay for the recycling of these tires.

For example in the Netherlands this is done by **RecyBEM**, for Belgium this would be **Recytyre** and in Ireland it's **Repak ELT**. A general process would look like this:



When tires are imported (or manufactured in a country) a recycling fee is paid for each tire to an organization, labeled as a collecting agency in

the illustration above. This collecting agency will later pay out to recycling companies when the tires are end-of-life.

1 <https://www.bna.com/beyond-bitcoin-blockchain-n73014476900/>

2 <https://www.allerin.com/blog/recycling-with-blockchain-how-cool-is-that>

3 <https://www.tudelft.nl/en/delft-outlook/articles/circularise-uses-blockchain-technology-to-trace-raw-materials/>

## 2. The issues: Complex bookkeeping and fraud

In these systems there are 2 key challenges that we would want to solve. The first challenge is the high administrative cost resulting from the way payments are structured. The second challenge is related to fraud due to grey import of tires that are offered to recycling companies.

By law it is the owner of the car who should pay the recycling fee and ideally that fee is deposited directly into a fund to prevent unnecessary transfers. Today these funds are transferred from entity to entity depending on how the tires are transferred through the value chain. This makes accountability easier from a follow-the-money

perspective but also has its disadvantages. First it requires each party to keep a record of all incoming and outgoing payments, second this method locks up expensive working capital as each party is obliged to finance the recycling fee upfront.

The other problem is fraud at the end of the chain. As there is no traceability of tires being sold more tires can be offered to recycling stations. This problem is outlined in the year report<sup>4</sup> of Recytyre, for the Belgian market 109,90% of the tires sold are offered for recycling, an excess of 7.543 tons.

## 3. Why and how blockchain can be applied

A blockchain is in fact a shared ledger that is distributed across multiple participants. This is a different architecture than what we mostly use today which is a centralized architecture. In a centralized architecture there is one controlling party that all other parties must trust. A centralized architecture aligns nicely with a lot of use cases that exist today such as a loyalty card scheme where the users that save points trust the organization to pay out on a later date. In cases where fraud amongst the parties is suspected this architecture could be challenged by some of the parties. This is also the case where value is exchanged between parties and where parties are incentivized to act dishonestly in order

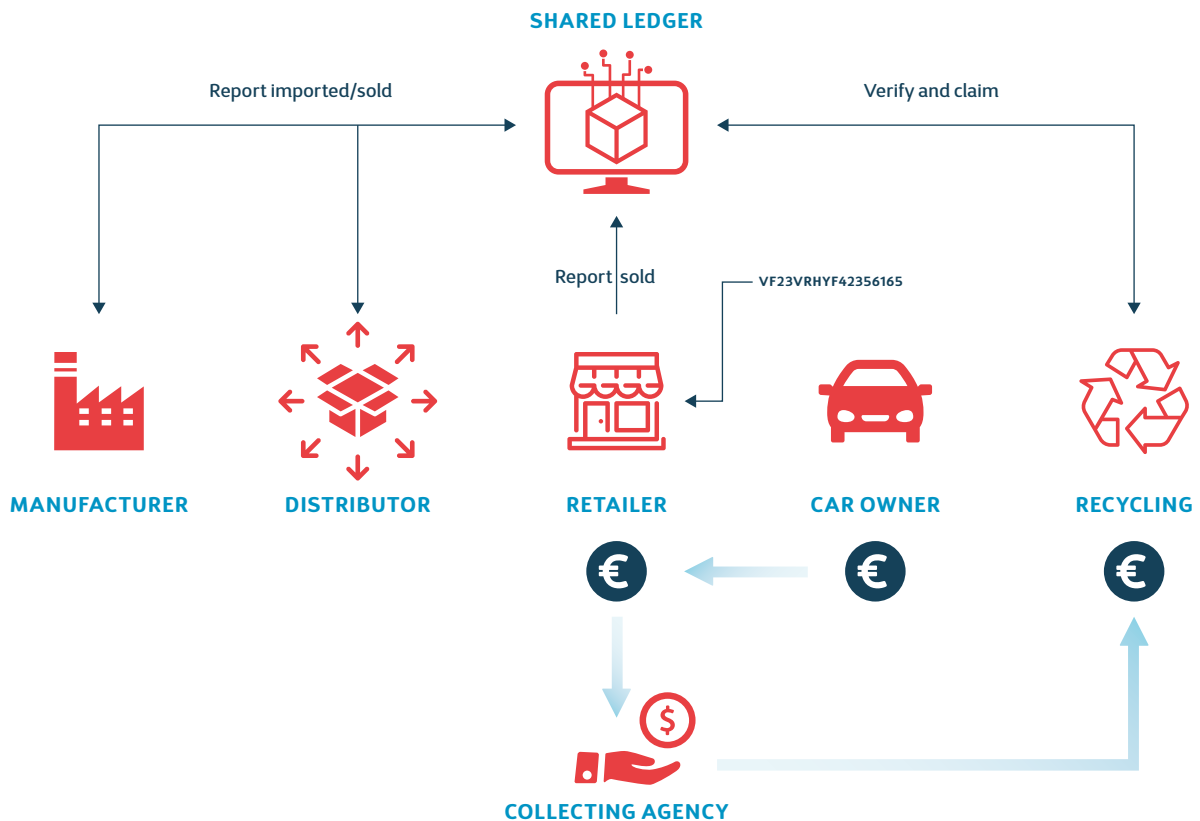
to protect themselves. In our case the recycle fee income stream is divided amongst the recycling parties based on the amount of tires they collect. If their competitors collect more tires from the grey market than they do they will receive a lower fee which will impact their financial results.

The other challenge we highlighted was that of high financing cost due to the obligation to pay all fees upfront. If there was a system that all parties could use to prove to the regulator that there were no unaccounted tires in the entire supply chain they could eliminate the reason for upfront financing and collect the fees at the Point of Sale when the tires are bought by a customer.

<sup>4</sup> <http://www.recytyre.be/media/89990/jaarverslag%202017%20recytyre%20-%20pdf.pdf>

To summarize a system that could make the internal accounting of recycled tires transparent to all parties could help to tackle fraud and

to prevent unnecessary upfront funding. Such a system could look like this:



In our alternative design we applied blockchain technology as a shared ledger for all parties. The new process flow would look like this:

1. Manufacturers report on the ledger how many tires they have produced for a country.
2. When tires are sold to a distributor both the manufacturer and the distributor confirm the sale on the ledger. This means that the manufacturer remains responsible for all tires he produced and can't allocate them without consent.
3. For the distributor the process is similar. When the tires are sold to the retailer both parties have to acknowledge the sale on the ledger.

These steps no longer require any upfront financing. The ledger can prove at any point how many tires a party holds. Tires that were "lost" will remain in the account of the party who lost them. It is possible to make tires expire automatically after a specific period of time, upon expiration the party that holds them is then automatically billed for the recycle fees.



The rest of the process looks like this:

4. When the retailer sells the tires to a consumer he marks the tires as sold on the ledger and logs the VIN number of the car on which the tires will be fitted. The retailer will also deposit the recycle fee with the collecting agency.
5. When the car owner wants to dispose of the tires he submits the VIN number to the recycling party. If there is no sale of tires to this VIN number on the ledger the owner has acquired these tires from i.e. another country and will need to pay the recycle fee.
6. The recycling company marks the tires as recycled and receives (a part of) the recycling fee from the collecting agency.

By registering the VIN number with the tires the loophole is closed and recycling parties can't claim recycling fees for tires for which they can't provide a VIN number. While we will use this approach for our use case and POC demonstration this approach is not fully "watertight" and we want to highlight some of the concerns that arose during our research:

#### Today tires are measured in weight, no by VIN number

When tires are collected and transported they are usually weighted and compensation is calculated based on weight. To facilitate this procedure we could look up the average weight of that tire and use that to exchange compensation between

collectors and recycling parties. Using this method implies shifting some of today's challenges to the "edge" of the supply chain which should be researched further to make sure it is an acceptable offer to make in comparison to the losses. Capturing and understanding the objections and benefits of all affected parties when introducing a new process is key in the adoption, or rejection, of new solutions.

#### Logging VIN numbers is time consuming/error prone for the retailer

To some it would make more sense to use a license plate as it's easier to memorize but a license plate is not always tied to a car, depending on which country you reside in. I.e. in The Netherlands a license plate is issued when a car is imported but in Belgium a license plate is issued to a person<sup>5</sup> and you can take the license plate off when you sell your car. A compromise could be sought in acquiring access to national license plate registries to do a lookup of the VIN number based on the license plate at that time.

#### It would be better to track individual tires

Tires have a DOT serial number<sup>6</sup> which could be used to individually register each tire. This would make the system completely foolproof but it is very time consuming to locate and enter each DOT number manually. RFID is currently research by the tire industry<sup>7</sup> which would solve some of the problems mentioned as the technology can be used to scan tires in bulk at a high speed.

<sup>5</sup> [https://en.wikipedia.org/wiki/Vehicle\\_registration\\_plates\\_of\\_Belgium](https://en.wikipedia.org/wiki/Vehicle_registration_plates_of_Belgium)

<sup>6</sup> <http://www.bridgestone.co.in/HowToReadTyreSideWall.aspx>

<sup>7</sup> <https://www.rubbernews.com/article/20180621/NEWS/180629982/column-rfid-is-transforming-the-tire-industry>

## 4. Building a blockchain business network with Hyperledger Fabric

Based on our research we found at least 2 use cases for the tire recycling industry that are worth exploring. Next we want to validate the following:

- Can blockchain technology be applied as outlined above?
- Can the expected benefits be realized?
- Are the processes feasible to implement?
- Did we miss anything? (either good or bad)
- What would be a logical next step for all parties?

In order to solicit constructive feedback from the various parties in the supply chain we will build an applications using our toolkit that will demonstrate the functionality that we intended. In the next chapter we'll demonstrate how to do this. For this application we will use the Hyperledger Fabric blockchain and if you follow the steps below you'll have it running in no time.

### 1. Install prerequisites

To run the template on your local machine you need to install Git and Docker.

- 1.1 Download and install Git for your OS from <https://git-scm.com/book/en/v2/Getting-Started-Installing-Git>. Git is used to clone the template project from our public repository on Github. On Windows you'll need to run this command as Administrator after installing: **git config --system core.longpaths true && git config --global core.eol lf && git config --global core.autocrlf input.**
- 1.2 Install Docker from <https://www.docker.com/get-started>. You'll need to create a (free) Docker community account. Docker is used to build and deploy your blockchain network. Start docker on your local machine and login to your account. Linux users will also need to install docker-compose.

### 2. Clone the template from the public Github repository

Clone the project in a folder on your local machine with the following command in a terminal: **git clone https://github.com/cegeka/hyperledger-recycling-poc**

When you have successfully copied the project, you can navigate into the newly created folder: **cd hyperledger-recycling-poc**


### 3. Start the blockchain network

Deployment is fully automated via a docker-compose container to initialize & deploy the blockchain network. Navigate to the docker folder (**cd docker**) and run the following command:

For MacOS and Linux: **sudo ./composer-setup.sh**

For Windows (run cmd as Administrator): **composer-setup.bat**

After about 15 minutes you can use your application by opening <http://localhost:8080> in your web browser. As a first step you can login as an administrator (user = "admin", password = "") to list all the users in your application (there are no passwords set for any of the users). To try our the use case, you can login as a manufacturer, distributor, retailer or recycling station. You can create a tire as a manufacturer and transfer it to a distributor, sell it to a retailer, put the tire on a car and recycle it as a last step.



**Build your  
own blockchain  
application**

---

Cegeka built the Hyperledger Composer Angular Quick-start template to speed up development of blockchain proof-of-concept projects. If you have a basic understanding of JavaScript and the Angular framework you will be able to clone our sample project and create your own blockchain application in no time.

This manual will help you to quickly start Hyperledger Composer projects and interact with them through a web frontend. It contains a basic backend (server folder) built on top of the standard composer-rest-server npm package with additional scripts to build & deploy the business network and a frontend application (client folder) with basic user management and transaction monitoring<sup>8</sup>. The project contains a fully self-contained Docker-compose deployment environment (docker folder) that can be used for production servers.

This manual does not cover the design of a business network. Excellent guides that cover this in great detail can be found at <https://hyperledger.github.io/composer/latest/tutorials/tutorials>.

We accept pull requests should you run into any problems and decide to fix them yourself. If you're keen to get started with Ethereum instead we highly recommend using Microsoft's Azure Blockchain Workbench solution found at <https://azure.microsoft.com/en-us/features/blockchain-workbench/>.

---

<sup>8</sup> The transaction monitoring (explorer) is currently disabled but will be fixed in Q1 2019

## 1. Create your own blockchain network

Before building your application you first need to create a business network to connect to the frontend application. Hyperledger Composer has overly simplified the process of building business networks. Even if you don't have programming skills, you can still create a simple business network with this manual.

Every business network that you built with Hyperledger consists of 3 base resources: Assets, Participants and Transactions. Simply put, a participant can interact with assets by writing a transaction to the ledger. Any use case you can think of can be modelled with these 3 base resources. If you need to make your model more complex you can write events and use enumerated types (enums) and concepts as a variable of a participant, asset or transaction. These resources are typically used in other object-oriented programming languages as well. To summarize:

- **Assets** can represent anything of value that can be shared or transacted
- **Participants** are the actors in a business network who make transactions.
- **Transactions** describe what can be done to the assets as they move around the business network

Just like with our example case you can get a blockchain application running right away, in this template you'll find very basic functionality in place where 3 parties can exchange assets.

If you want to get started right away and design your own blockchain network go to <https://composer-playground.mybluemix.net/> and use the online playground of Hyperledger Composer (free) to create your own business network definition or analyze one of the provided samples. Hyperledger Composer uses an object-oriented modeling language to define the domain model and to implement access control rules in your business network. If you're happy with your business network you can easily copy-and-paste your code in this template to transform it into a real application with a web frontend by following the instructions below.

## 1. Install prerequisites

To run the template on your local machine you first need to install Git and Docker. If you haven't already done so please refer to the previous chapter for detailed steps.

## 2. Clone the template from the public Github repository

Find the code base in the following public Github repository. This is a slightly different repository than our example case in the previous chapter so you can start off fresh:

<https://github.com/cegeka/hyperledger-poc-quickstart>

Clone the project in a folder on your local machine with the following command in a terminal:

**git clone** <https://github.com/cegeka/hyperledger-poc-quickstart.git>

When you have successfully copied the project, you can change directory in your terminal to move into the newly created folder: **cd hyperledger-poc-quickstart**

## 3. Editing the Hyperledger Composer code (optional)

If you have designed your own business network you can copy-paste your code into the corresponding files below. If not you can skip right to step 5 and run the included application.

**Data model:** `server/models/com.cegeka.cto`

**Access control rules:** `server/permissions.acl`

**Transaction logic:** `server/lib/logic.js`

**Named Queries:** `server/queries.qry`

You can use Visual Studio Code (free) or any other integrated development environment to update the Hyperledger Composer files in your project: <https://code.visualstudio.com/>

*Note: Before a business network definition can be deployed all the information must be packaged into a Business Network Archive (.bna) file. This file is automatically created and deployed in the template.*

## 4. Change composer data initialization script (optional)

The script in `server/setup/setup.js` runs after the composer network is started. It will create all pre-registered composer entities, like user accounts & initial assets. Customize the JavaScript file to create all entities by calling the appropriate REST endpoints.

Update the setup script with your own user functions to initialize participants from your data model file. The function `createCustomer` initializes the demo application with 3 participants of type `Customer`.

```
30 /**
31  * Create customer user function
32  */
33 function createCustomer(customerId, password, firstName, lastName)
34     var options = {
35         method: 'POST',
36         uri: rootUrl + 'Customer',
37         body: {
38             customerId: customerId,
39             password: password,
40             firstName: firstName,
41             lastName: lastName
42         },
43         json: true
44     };
45
46     request(options).then(function(response) {
47         console.log('createCustomer ' + customerId, response);
48     }, function(err) {
49         console.error('createCustomer ' + customerId, err);
50     });
51 }
52
53 /**
54  * Execute functions
55  */
56 createAdmin();
57 createCustomer('customer1', '', 'First', 'Customer');
58 createCustomer('customer2', '', 'Second', 'Customer');
59 createCustomer('customer3', '', 'Third', 'Customer');
60
```

## 5. Start the blockchain network

The following script has to be ran every time the Hyperledger Composer code is changed. It will take care of rebuilding the Hyperledger composer backend, start the Fabric network, install the composer application and initialize the sample data.

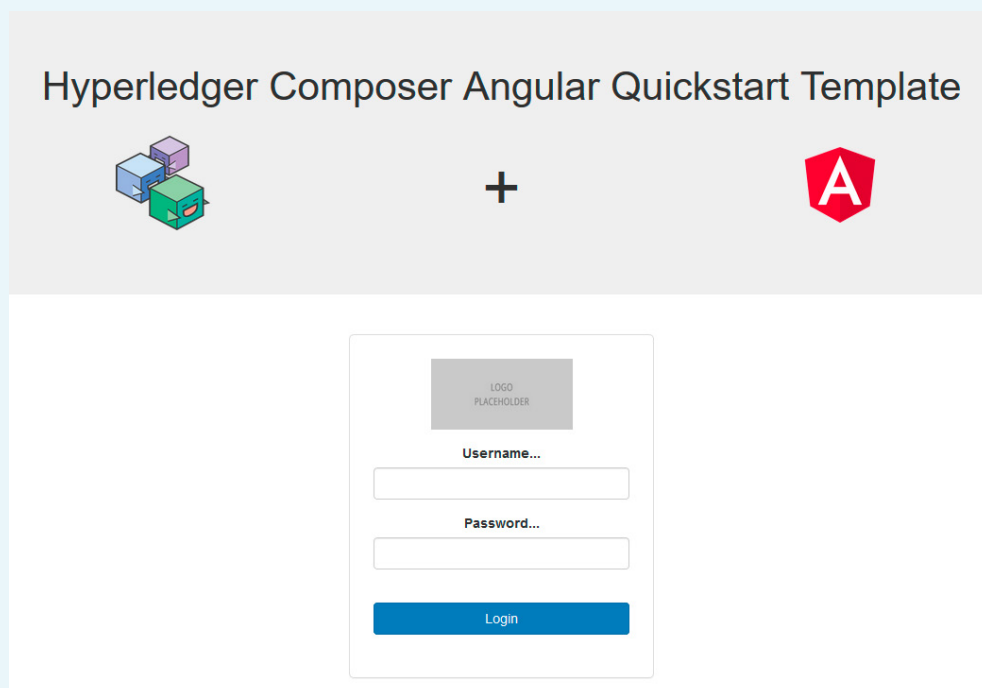
Deployment is fully automated via a docker-compose container to initialize & deploy the blockchain network. Navigate to the docker folder (**cd docker**) and run the following command:

For MacOS and Linux: **sudo ./composer-setup.sh**

For Windows (run cmd as Administrator): **composer-setup.bat**



Well done, now you can use your application! Login with the admin role to manage the users in your application and create functionality for your app's users. By default, the frontend will connect to <http://localhost:8080> and redirect to the login page. In the demo, you can login with username: admin, customer1, customer2, customer3 (all without passwords). You can create new participants of the class Customer with the admin user.



## 6. Customize frontend application (optional)

You can customize the frontend application and connect it with your own blockchain network. Web applications that need to interact with a deployed business network should make calls to a REST API.

A connection with the REST API is already implemented in the template. Using our demo Angular project as example, you can copy pieces of code to create the logic that is needed to interact with the participants, assets and transactions you've created. Once you understand how to use the GET, POST, PUT and DELETE actions of the REST API, you will have your own customized application up and running really fast.

If you completed step 5 you can operate all REST API actions in the Hyperledger Composer REST server which is available at <http://localhost:3000/explorer/>

You can customize most aspects of your frontend application such as your logo or the colors of the banner on the login page and top menu for logged in users. After that you can add the participants of your business network as user roles in the application's user service, create pages for your participants, manage login redirects, implement assets and transactions, use transaction monitoring to show specific transaction details in the blockchain transaction detail view. Finally, you should also create menu items based on the user roles in your application. The steps below detail these options step by step:

- 7.1 Update the logo placeholder in both the login form in the login page (client/src/app/pages/login/login.component.html) and in the top navbar menu items in the banner page (client/src/app/components/banner.component.html). You can add an image of your choice to the folder (client/src/app/assets) which can be used to replace the placeholder images. You can also simply remove the placeholder image from the assets folder, and add your logo with exact the same title 'logo-placeholder.jpg'.
- 7.2 Update the colors of the banner on the login page and top menu for logged in users by changing the background color of .navbar-default in the css styles component (client/src/app/components/banner/banner.component.css) and .banner in the main css styles file (client/src/styles.css).
- 7.3 Edit the UserRole enum in client/src/app/services/user.service.ts and add the roles used by the application. The string value of the role must match the Hyperledger Composer object name in the DTO. Also edit the getUserRole() method to infer the role based on user names, or replace with a different mechanism.

```
72         return userRole;
73     });
74 }
75
76 public login(userName: string, password: string): Observable<UserRole> {
77     const userRole = this.getUserRole(userName);
78
79     return this.loginWithRole(userName, userRole, password);
80 }
81
82 private getUserRole(userName): UserRole {
83     if (!userName) {
84         return null;
85     }
86     // TODO: figure out on which user class to check, based on user name
87
88     if (userName.indexOf('customer') !== -1) {
89         return UserRole.Customer;
90     }
91
92     return UserRole.Admin;
93 }
94 }
95
96 //
97 // Enum with the types of user classes supported by the application.
98 // The string value maps to the URL generated by the composer rest API tool
99 export enum UserRole {
100     Admin = 'Admin',
101     Customer = 'Customer'
102 };
103
```

#### 7.4 Create any needed home pages for the new roles and add them to the routes list (client/src/app/app.routes.rs) and modules list (client/src/app/app.modules.ts)

```

1 import { LoginComponent } from "../pages/login/login.component";
2 import { HomeComponent } from "../pages/admin/admin-home/home.component";
3 import { CustomerComponent } from "../pages/customer/customer-home/customer.component";
4 import { AccountComponent } from "../pages/customer/customer-account/account.component";
5 import { AssetsComponent } from "../pages/customer/customer-assets/assets.component";
6 import { BlockchainComponent } from "../pages/blockchain/blockchain.component";
7 import { TxDetailComponent } from "../pages/tx-detail/tx-detail.component";
8
9 export const routes = [
10   {
11     path: '',
12     component: LoginComponent
13   },
14   {
15     path: 'login',
16     component: LoginComponent
17   },
18   {
19     path: 'admin',
20     component: HomeComponent,
21   },
22   {
23     path: 'customer',
24     component: CustomerComponent
25   },
26 ]

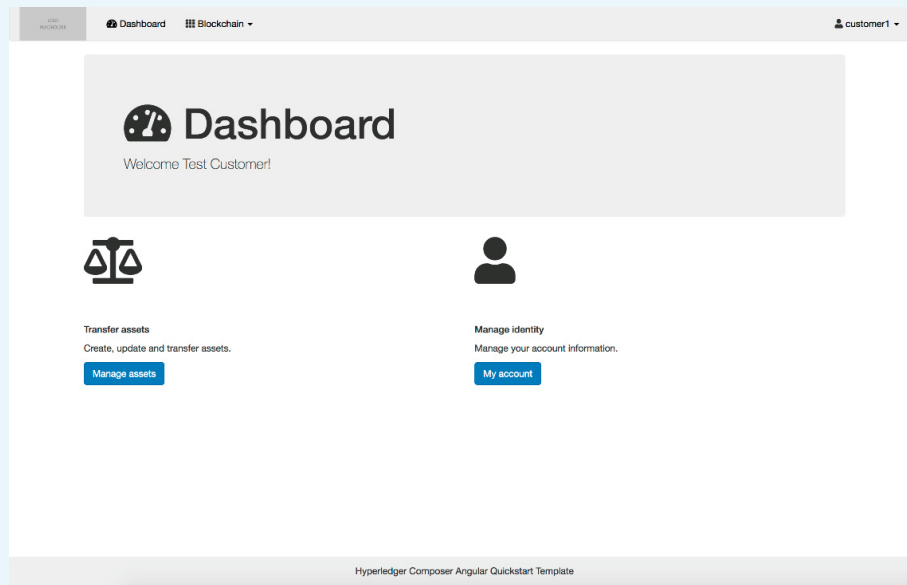
```

```

1 import { BrowserModule } from '@angular/platform-browser';
2 import { CommonModule } from '@angular/common';
3 import { HttpClientModule } from '@angular/http';
4 import { RouterModule } from '@angular/router';
5 import { NgModule } from '@angular/core';
6 import { FormsModule } from '@angular/forms';
7 import { BootstrapModalModule } from 'ng2-bootstrap-modal';
8
9 import { AppComponent } from './app.component';
10 import { routes } from './app.routes';
11
12 import { FilterPipe } from './pipes/filter.pipe';
13
14 import { LoginComponent } from './pages/login/login.component';
15 import { HomeComponent } from './pages/admin/admin-home/home.component';
16 import { CustomerComponent } from './pages/customer/customer-home/customer.component';
17 import { AssetsComponent } from './pages/customer/customer-assets/assets.component';
18 import { AccountComponent } from './pages/customer/customer-account/account.component';
19 import { BlockchainComponent } from './pages/blockchain/blockchain.component';
20 import { TxDetailComponent } from './pages/tx-detail/tx-detail.component';
21
22 import { UserService } from './services/user.service';
23 import { AssetService } from './services/asset.service';
24 import { HistoryService } from './services/history.service';
25
26 import { BannerComponent } from './components/banner/banner.component';

```

There's a dashboard as homepage for the customer role by creating the Customer component (client/src/app/pages/customer/customer-home/customer.component.ts) and the html content for the page (client/src/app/pages/customer/customer-home/customer.component.html).



- 7.5 Alter the way login redirects work, based on user roles. Change the method login() in the client/src/app/pages/login/login.components.ts class to perform redirect to the correct user-role specific routes.

```

1  import { Component, OnInit } from '@angular/core';
2  import { Router } from '@angular/router';
3  import { UserService, UserRole } from '../../services/user.service';
4
5  @Component({
6    selector: 'app-login',
7    templateUrl: './login.component.html',
8    styleUrls: ['./login.component.css']
9  })
10 export class LoginComponent implements OnInit {
11   error: string;
12
13   constructor(private router: Router, private userService: UserService) { }
14
15   ngOnInit() {
16     this.userService.cleanup();
17   }
18
19   login(user: string, password: string) {
20     this.error = '';
21     this.userService.login(user, password).subscribe((role) => {
22       if (role === UserRole.Customer) {
23         this.router.navigate(['/customer']);
24       } else if (role === UserRole.Admin) {
25         this.router.navigate(['/admin']);
26       }
27     },
28     (error) => {
29       if (typeof(error) === 'string') {
30         this.error = error
31       } else {
32         this.error = "Login failed";
33       }
34       console.error('Login failed', error);
35     });
36   }
37
38 }
39

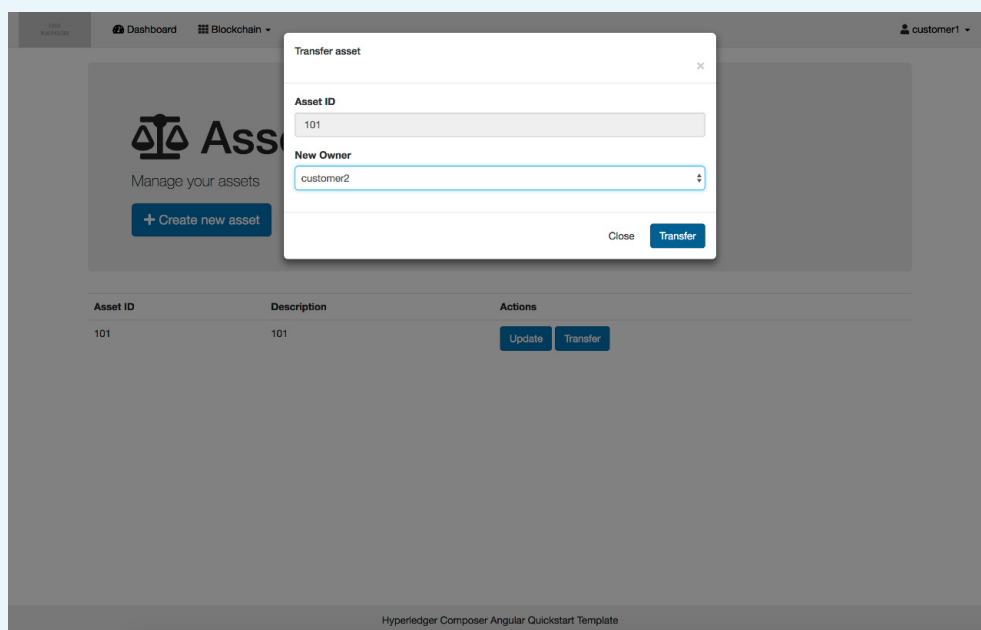
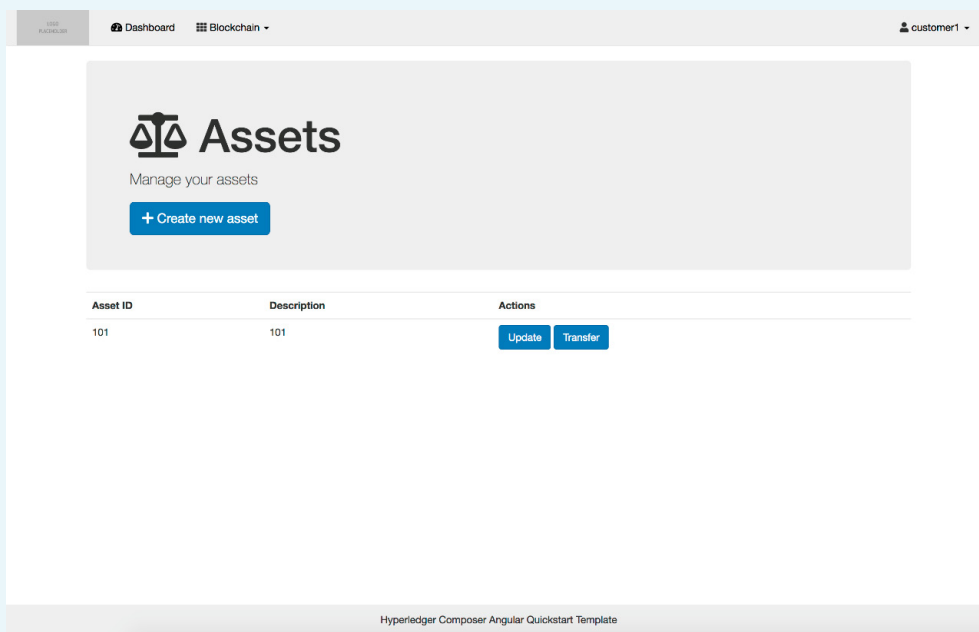
```

A sample asset transferring demo application is part of the template, which shows you how to implement an asset Tradeable and a transaction Trade for the user role Customer. In file `asset.service.ts` the GET, POST, PUT and DELETE actions are created for the sample asset Tradeable. The `tradeCommodity` function is the POST action in this service component for the Trade transaction.

- 7.6 Add a new service for assets in your blockchain network by extending the `BaseResourceService` class (take a look at `client/src/app/services/asset.service.ts`). Register these services in the `client/src/app/app.modules.ts` class.

```
1 import { Router } from '@angular/router';...
9
10 @Injectable()
11 export class AssetService extends BaseResourceService {
12
13   getTradeables(): Observable<any> {
14     return this.jsonRequest('Tradeable', HTTP_VERB.GET);
15   }
16
17   createTradeable(assetId: string, description: string, owner: string): Observable<any> {
18     return this.jsonRequest('Tradeable', HTTP_VERB.POST, {
19       assetId: assetId,
20       description: description,
21       owner: owner
22     });
23   }
24
25   deleteTradeable(id: string): Observable<any> {
26     return this.jsonRequest('Tradeable/${id}', HTTP_VERB.DELETE, {
27       id: id
28     });
29   }
30
31   updateTradeable(id: string, description: string, owner: string): Observable<any> {
32     return this.jsonRequest('Tradeable/${id}', HTTP_VERB.PUT, {
33       id: id,
34       description: description,
35       owner: owner
36     });
37   }
38
39   tradeCommodity(tradeable: string, sender: string, newOwner: string): Observable<any> {
40     return this.jsonRequest('Trade', HTTP_VERB.POST, {
41       tradeable: tradeable,
42       sender: sender,
43       newOwner: newOwner
44     });
45   }
46
47 }
48
```

The logic for the Tradeable asset and Trade transaction is implemented in the `assets.component.ts` file (`client/src/app/pages/customer/customer-assets/assets.component.ts`). The file `assets.component.html` demonstrates how you can implement a table to list your assets with the functionality to create, update and delete assets. The `tradeCommodity` function shows how the sample transaction `Trade` can be used by the user role `Customer`. The button 'Transfer' opens a modal (pop up screen) on which a user can transfer assets to other users of the `Customer` class.



## 7.7 Add a sample asset and transaction

When you have created the `asset.service.ts` file in the previous step, then you can use the service to implement the GET, POST, PUT and DELETE actions for assets and the POST action for transactions in the frontend application.

The standard REST API actions can be used to create, update and delete assets and participants in the ledger. But these functions of the REST API don't allow you to add specific transaction details in the frontend application. Therefore, a better solution is to create specific transactions for creating, updating and deleting asset and participant objects. The Trade transaction demonstrates how this works. First in the file `history.service.ts` the `getAssetTransferredTx` function is created to retrieve all Trade transactions.

```

1 import { Observable } from "rxjs/Observable";
2 import { Injectable } from '@angular/core';
3 import { BaseResourceService, HTTP_VERB } from './base-resource.service';
4
5 @Injectable()
6 export class HistoryService extends BaseResourceService {
7
8   getBlockchainOverview(): Observable<any> {
9     return this.jsonRequest('system/historian', HTTP_VERB.GET);
10  }
11
12   getTx(transactionId: string): Observable<any> {
13     return this.jsonRequest('system/historian/${transactionId}', HTTP_VERB.GET);
14  }
15
16   getAssetTransferredTx(transactionId: string): Observable<any> {
17     return this.jsonRequest('Trade/${transactionId}', HTTP_VERB.GET);
18  }
19 }
20

```

In the file `tx-detail.component.ts` (`client/src/app/pages/tx-detail/tx-detail.component.ts`) the `addSpecificTx` function is updated with the Trade class, and the `addAssetTransferred` functions imports all the fields of transaction type Trade.

```

42 // Support for custom transaction types
43 addSpecificTx() {
44   for (let jsonItem of this.json) {
45     if (jsonItem.key == this.TransactionTypeKey) {
46       if (jsonItem.value.indexOf('Trade') != -1) {
47         jsonItem.value = 'Asset transferred';
48         this.addAssetTransferred();
49       }
50     }
51   }
52 }
53
54 addAssetTransferred() {
55   this.historyService.getAssetTransferredTx(this.id).subscribe(this.addAllFields.bind(this));
56 }
57
58 private addAllFields(response) {
59   for (let key in response) {
60     if (!this.jsonContainsInfo(key)) {
61       this.json.push({ key: key, value: response[key] });
62     }
63   }
64 }
65
66 private jsonContainsInfo(key: string) {
67   for (let item of this.json) {
68     if (item.key == key) {
69       return true;
70     }
71   }
72   return false;
73 }
74 }
75

```



In the tx-detail.component.html file you can update the view of the component. Here you can see how the details of the Trade transactions are shown on the transaction detail page. In this example, we show the transaction type, asset type, sender, new owner and the timestamp of the transaction. Add your own ng-container tags in the table and customize the \*ngIf statements with the transaction detail items that you want to show.

```

20     <!-- Add specific transaction details -->
21     <ng-container *ngIf="item.key=='transactionType'">
22         <td>Transaction Type</td>
23         <td>{{item.value}}</td>
24     </ng-container>
25
26     <ng-container *ngIf="item.key=='tradeable'">
27         <td>Asset</td>
28         <td>{{item.value.split('.')[2]}}</td>
29     </ng-container>
30
31     <ng-container *ngIf="item.key=='sender'">
32         <td>Sender</td>
33         <td>{{item.value}}</td>
34     </ng-container>
35
36     <ng-container *ngIf="item.key=='newOwner'">
37         <td>New Owner</td>
38         <td>{{item.value}}</td>
39     </ng-container>
40
41     <!-- Generic stuff -->
42     <ng-container *ngIf="item.key=='transactionTimestamp'">
43         <td>Timestamp</td>
44         <td>{{item.value | date:'dd MMM yyyy, HH:mm:ss'}}</td>
45     </ng-container>
46 </tr>
47 </table>

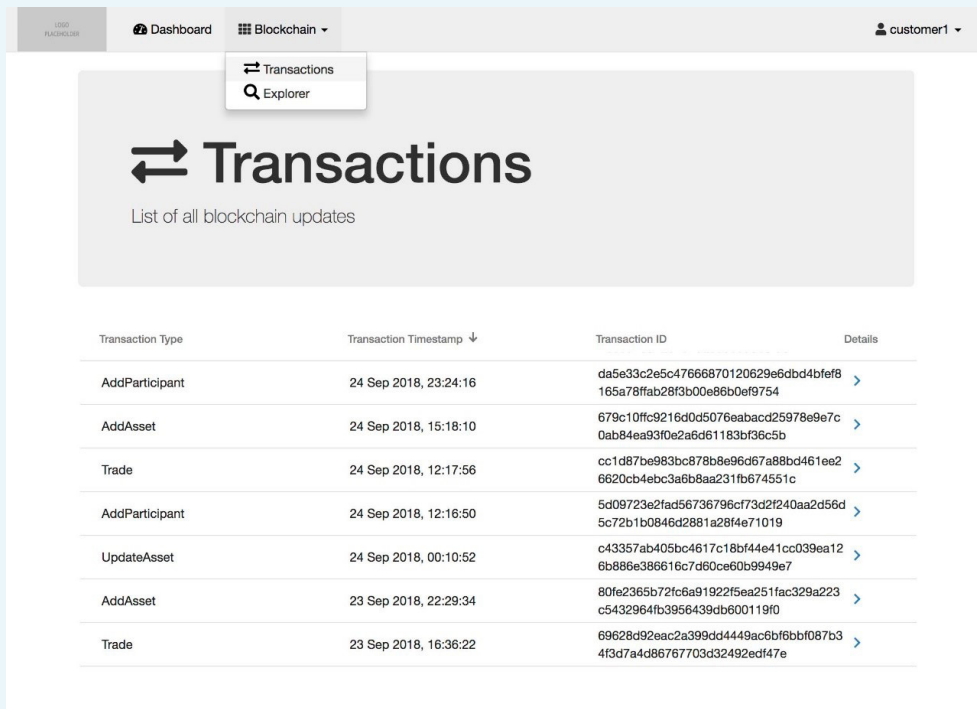
```

The result is the tx-detail page where there are specific transaction details for each individual Trade transaction in the ledger. Go to the top menu in your application, click on blockchain and choose the sub item Transactions. In this list, you see all ledger updates from the start of your successful deployment of the business network. Click on the arrow at the right side of the table rows to see transaction details of transactions.



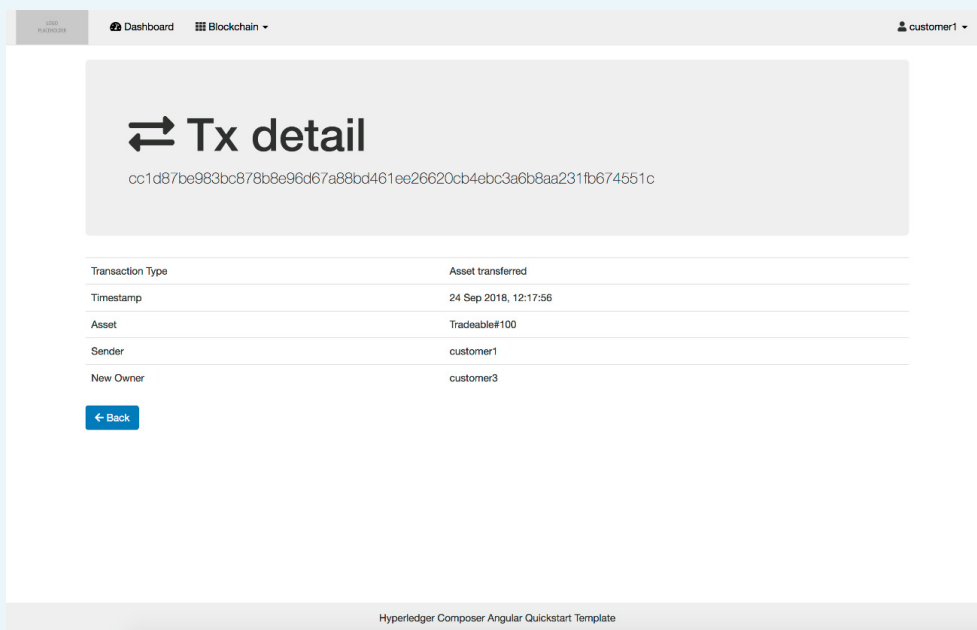
## 7.8 Implement custom transaction monitoring

The `client/src/app/pages/tx-detail/tx-detail.component.ts` class is used to display details about specific transactions. This class (and the underlying `history.service.ts` file) can be customized to retrieve additional business information for specific transaction types and present this to the user.



The screenshot shows a web application interface for monitoring transactions. The top navigation bar includes 'Dashboard' and 'Blockchain' menus, and a user profile 'customer1'. A dropdown menu is open over the 'Transactions' header, showing options for 'Transactions' and 'Explorer'. The main content area features a large heading 'Transactions' with a double-headed arrow icon and the subtitle 'List of all blockchain updates'. Below this is a table listing transactions.

Transaction Type	Transaction Timestamp ↓	Transaction ID	Details
AddParticipant	24 Sep 2018, 23:24:16	da5e33c2e5c47666870120629e6dbd4bfe8165a78ffab28f3b00e86b0ef9754	>
AddAsset	24 Sep 2018, 15:18:10	679c10ffc9216d0d5076eabacd25978e9e7c0ab84ea93f0e2a6d61183bf36c5b	>
Trade	24 Sep 2018, 12:17:56	cc1d87be983bc878b8e96d67a88bd461ee26620cb4ebc3a6b8aa231fb674551c	>
AddParticipant	24 Sep 2018, 12:16:50	5d09723e2fad56736796cf73d2f240aa2d56d5c72b1b0846d2881a28f4e71019	>
UpdateAsset	24 Sep 2018, 00:10:52	c43357ab405bc4617c18bf44e41cc039ea126b886e386616c7d60ce60b9949e7	>
AddAsset	23 Sep 2018, 22:29:34	80fe2365b72fc6a91922f5ea251fac329a223c5432964fb3956439db60011910	>
Trade	23 Sep 2018, 16:36:22	69628d92eac2a399dd4449ac6bf6bbf087b34f3d7a4d86767703d32492edf47e	>



The screenshot shows the 'Tx detail' page for a specific transaction. The page features a large heading 'Tx detail' with a double-headed arrow icon and the transaction ID `cc1d87be983bc878b8e96d67a88bd461ee26620cb4ebc3a6b8aa231fb674551c`. Below this is a table with transaction details.

Transaction Type	Asset transferred
Timestamp	24 Sep 2018, 12:17:56
Asset	Tradeable#100
Sender	customer1
New Owner	customer3

A blue 'Back' button is located at the bottom left of the page.

Hyperledger Composer Angular Quickstart Template

## 7.9 Implement menu items based on the user roles in your application.

Update the routerlinks in the top menu navbar in the Banner component html file (client/src/app/components/banner/banner.component.html).

```
1 <!-- Responsive top navbar for loggedin users -->
2 <nav class="navbar navbar-default">
3   <div class="container-fluid">
4     <div class="navbar-header">
5       <button type="button" class="navbar-toggle collapsed" data-toggle="collapse" data-target="#navbar-collapse-1"
6         aria-expanded="false">
7         <span class="sr-only">Toggle navigation</span>
8         <span class="icon-bar"></span>
9         <span class="icon-bar"></span>
10        <span class="icon-bar"></span>
11      </button>
12      <!-- Create logo routerlink for different user types based on loggedInRole -->
13      <a class="navbar-brand logo" [routerLink]="['/admin']" *ngIf="userService.loggedInRole == UserRole.Admin">
14        </a>
15      <a class="navbar-brand logo" [routerLink]="['/customer']" *ngIf="userService.loggedInRole == UserRole.Customer">
16        </a>
17      <a class="navbar-brand logo" [routerLink]="['/']" *ngIf="!userService.loggedInRole">
18        </a>
19    </div>
```

## 2. About support

Cegeka does not offer any support on these repositories but we do maintain them. If you run into any issues you can open a ticket on Github and we will respond to your request as soon as possible.



Cegeka has experimented with and worked on blockchain projects since 2015 and is considered a pioneer in Europe. Thanks to our business partners Cegeka can offer a comprehensive blockchain solution from idea to hosting, maintenance and support.

**Visit [www.cegeka.com](http://www.cegeka.com) for more information.**

**HEADQUARTERS:**

Universiteitslaan 9  
3500 Hasselt  
Belgium

**MORE OFFICES:**

[www.cegeka.com/offices](http://www.cegeka.com/offices)

**FOLLOW US ON**



[info@cegeka.be](mailto:info@cegeka.be)



[www.cegeka.com](http://www.cegeka.com)



[www.linkedin.com/company/cegeka](http://www.linkedin.com/company/cegeka)



[www.twitter.com/cegeka](http://www.twitter.com/cegeka)



[www.facebook.com/cegeka](http://www.facebook.com/cegeka)

---

**[WWW.CEGEKA.COM](http://WWW.CEGEKA.COM)**