Proof of concept framework release

# Introduction

Create your own blockchain application with Cegeka’s open source proof of concept framework. This document explains how you can create a blockchain network, how to customize a frontend application and how to deploy your app in the cloud to make it publically available. Each chapter starts with an explanation and provides a list of actions you should perform to use the template. You don’t have to bother about complex multistep deployment processes, because we automated this for you!

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 programming language JavaScript and the Angular framework, then you will be able to clone our sample project and create your own blockchain app quite easily. But even if you can’t program you will be able to copy our demo application and host your own blockchain application on Microsoft Azure.

This template is meant as a quick-start for Hyperledger Composer projects. 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 blockchain network and a frontend application (client folder) with basic user management and transaction monitoring. The project contains a fully self-contained Docker-compose deployment environment (docker folder) that can be used for production servers.

Please keep in mind that the framework is meant to be used in the proof of concept phase of projects. We strongly advise not to bring any application that is built on top of this template to production. The template is mainly used to experiment with blockchain technology and to build applications for demo purposes. Though, it is a good way to learn how you can build blockchain applications with intuitive Hyperledger Composer development tools.

## 1.1 Action: Install prerequisites

To run the template on your local machine you first need to install Git, Docker, Nodejs and Angular CLI.

1.1.1 Download and install Git 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 (action 1.2).

1.1.2 Download and install Docker from <https://www.docker.com/get-started> and create a free docker community account. Docker is used to build and deploy your blockchain network (action 2.3). Start docker on your local machine and login to your account.

1.1.3 Download and install Nodejs from <https://www.npmjs.com/get-npm>. Node package manager (npm) which comes with Nodejs is needed to install the dependencies (action 3.1) of the frontend application on your local machine.

1.1.4 Download and install Angular CLI by installing it globally using a terminal. Enter the command: **npm install -g @angular/cli**. The Angular Command Line Interface (CLI) is needed to run the frontend application (action 3.1) on your local machine.

## 1.2 Action: Clone the template from the public Github repository

Find the code base in the following public Github repository:

<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**](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**

# Create your own blockchain network

Before building your app, you first need to create a working blockchain network to connect to the frontend application. Hyperledger Composer has overly simplified the process of building blockchain networks. Even if you don’t have programming skills, you would still be able to create (parts of) a simple blockchain network. In the documentation of Hyperledger Composer you can read that both business analysts and programmers can be involved in creating a blockchain network.

Go to <https://composer-playground.mybluemix.net/> and use the online playground of Hyperledger Composer free of charge to create your own business network definition or analyze one of the provided samples to get a feeling with the tooling.

Hyperledger Composer includes an object-oriented modeling language that is used to define the domain model and to implement access control rules in your business network definition.

**The model file**

Every business network consists of the base resources: Assets, Participants and Transactions. Simply said, 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 want to make your model more complex, then 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.

* **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.

**The script file:** The scripts for a business network definition are written in JavaScript and refer to the types that are defined in the model file.

**Access Control file:** The Access Control file contains a set of access control rules that define the rights of the different participants in the business network.

**Query file:** Queries are used to return data about the blockchain world-state. Queries are an optional component of a business network definition, written in a single query file.

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

*Skip action 2.1 and 2.2 if you want to start the demo network that is included in the template in action 2.3 without the need to write any code.*

## 2.1 Action: Add the Hyperledger Composer code

Update the following Hyperledger Composer files with the proper business definition:

**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 (for free) or any other integrated development environment to update the Hyperledger Composer files in your project: <https://code.visualstudio.com/>

Read the tutorials in the online documentation of Hyperledger Composer if you want to learn more: <https://hyperledger.github.io/composer/latest/tutorials/tutorials>

## 2.2 Action: Change composer data initialization script

The script in server/setup/setup.js is meant to be ran 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.



## 2.3 Action: 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.

First move to the docker folder (cd docker) by changing directory from the main folder to the docker folder using your terminal. Run the following command in the docker folder to build all docker containers:

**docker-compose build**

Deployment is automated via a docker-compose container to initialize & deploy the blockchain network. Run the following command in the docker folder as a superuser (required on Linux to delete the existing docker-compose volumes):

**sudo ./composer-setup.sh**

*The first time that you run this command it fails in the current setup of the template, but the second time it succeeds. Once the script is finished, your network will be running and is ready to be used.*

# Customize frontend application

In this section, we explain how you can customize the frontend application and how to 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 action 2.3 Start the blockchain network, you can operate all REST API actions in the Hyperledger Composer REST server which is available at <http://localhost:3000/explorer/>

## 3.1 Action: Start the frontend application

If you are still in the docker folder in your terminal, first move up one directory (cd ..) back to the main folder and then change directory again to the client folder (cd client).

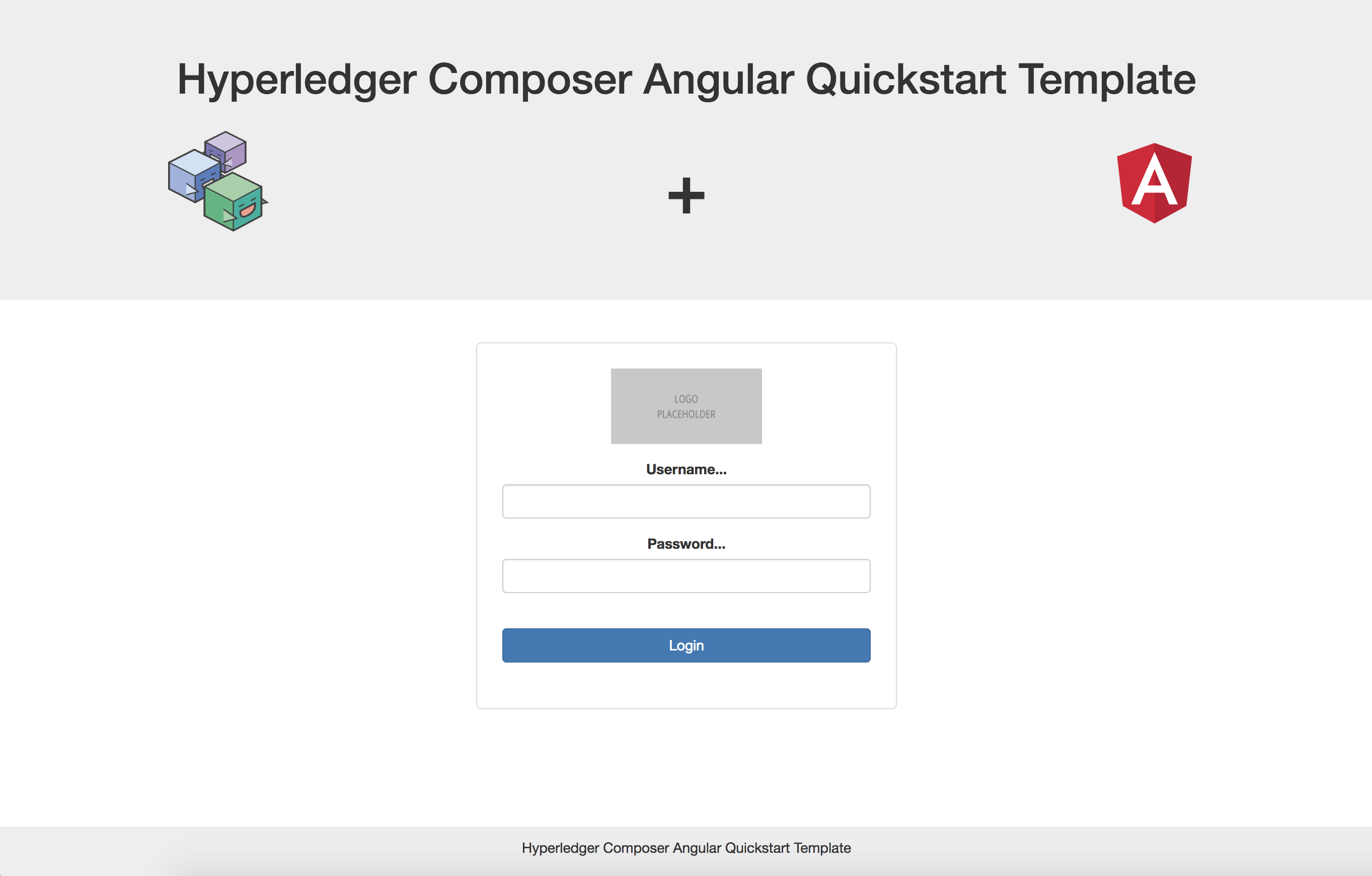
Install the frontend application dependencies in the client folder:

**npm install**

Start the frontend application development server with Angular CLI in the client folder:

**ng serve**

Well done, now you can use your app! 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:4200> and redirect to the login page. In the demo, you can login with username: admin, customer1, customer2, customer3. You can create new participants of the class Customer with the admin user.



Start customizing the frontend application with some basic styles for the app such as a logo or the colors of the banner on the login page and top menu for logged in users. After that, add the participants of your blockchain model 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.

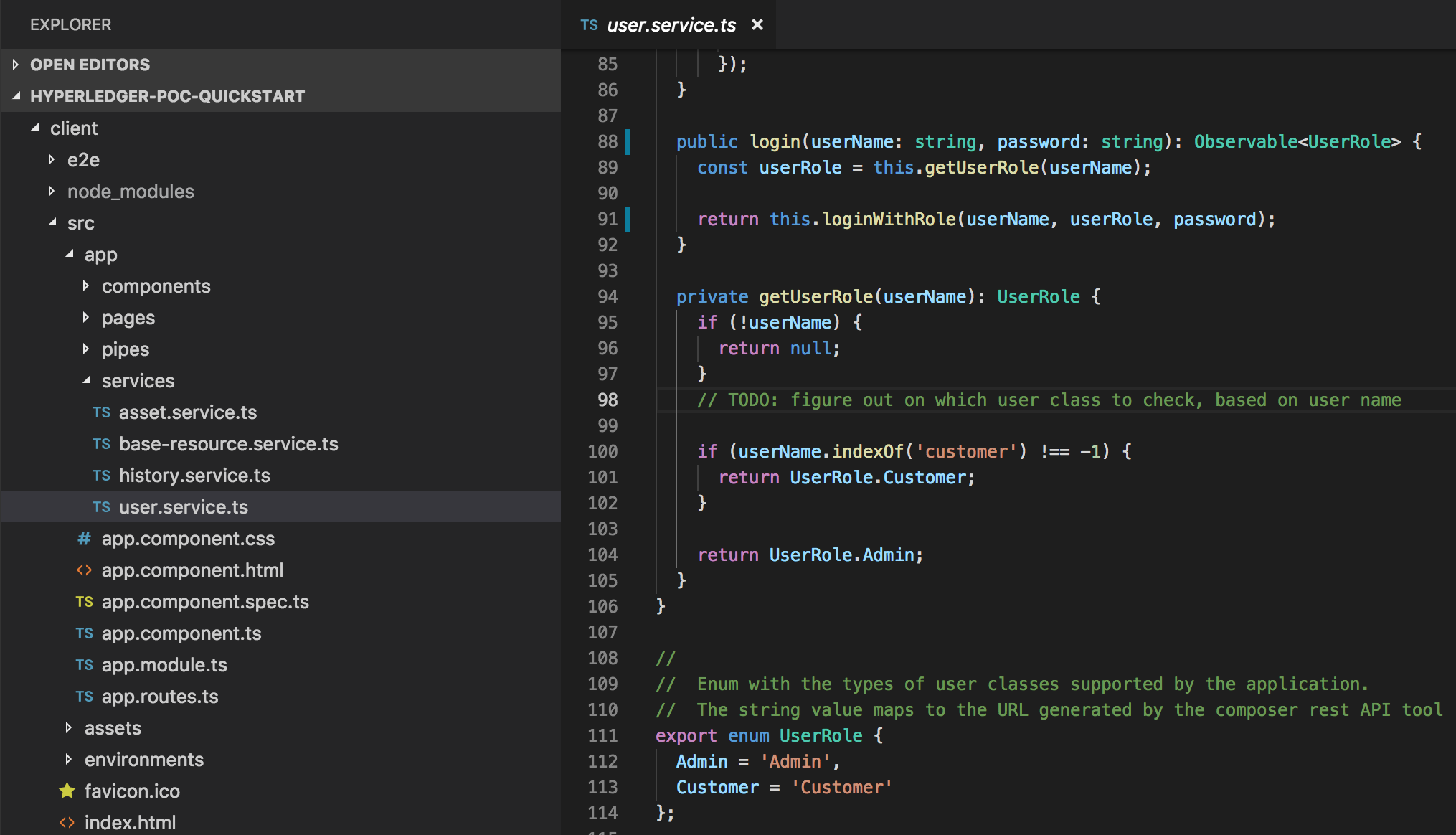
## 3.2 Action: Customize frontend application

3.2.1 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’.

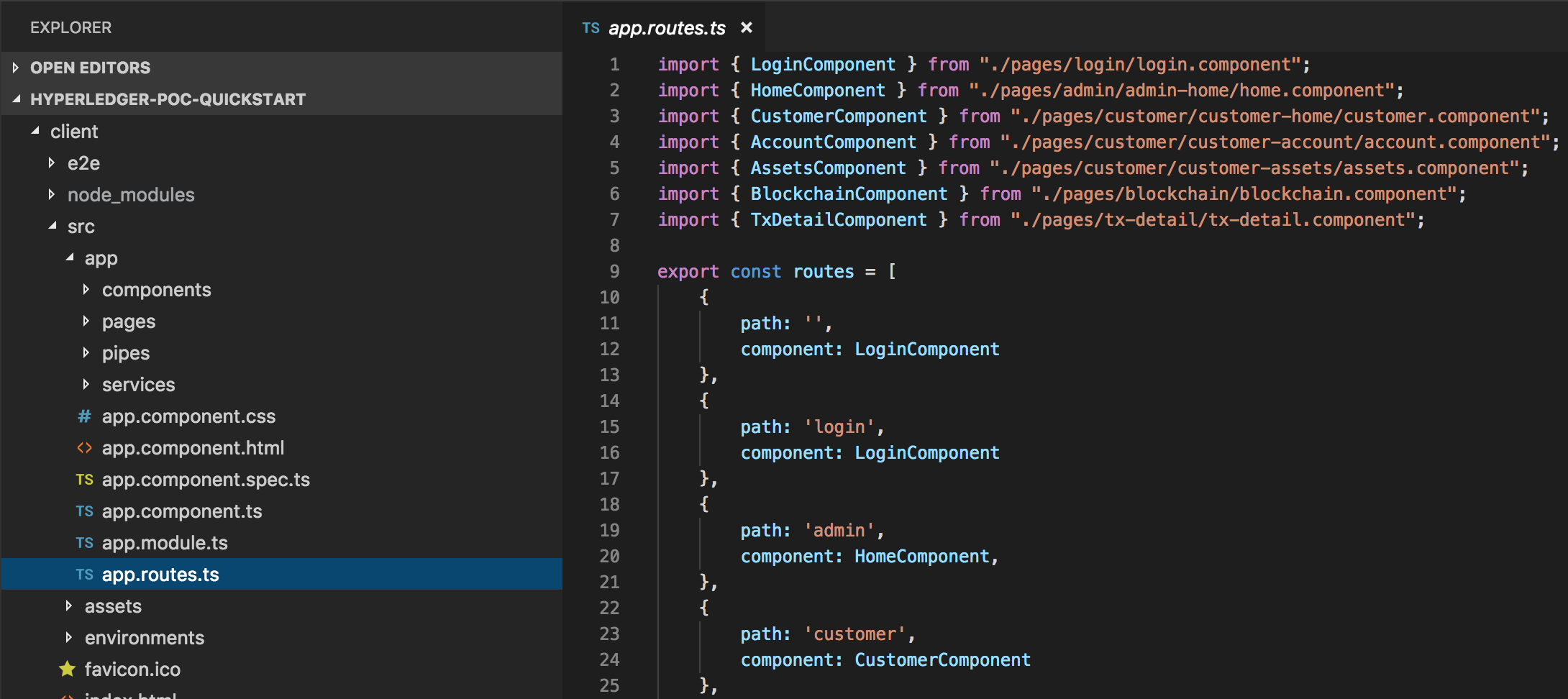
3.2.2 Update the colors of the banner on the login page and top menu for logged in users

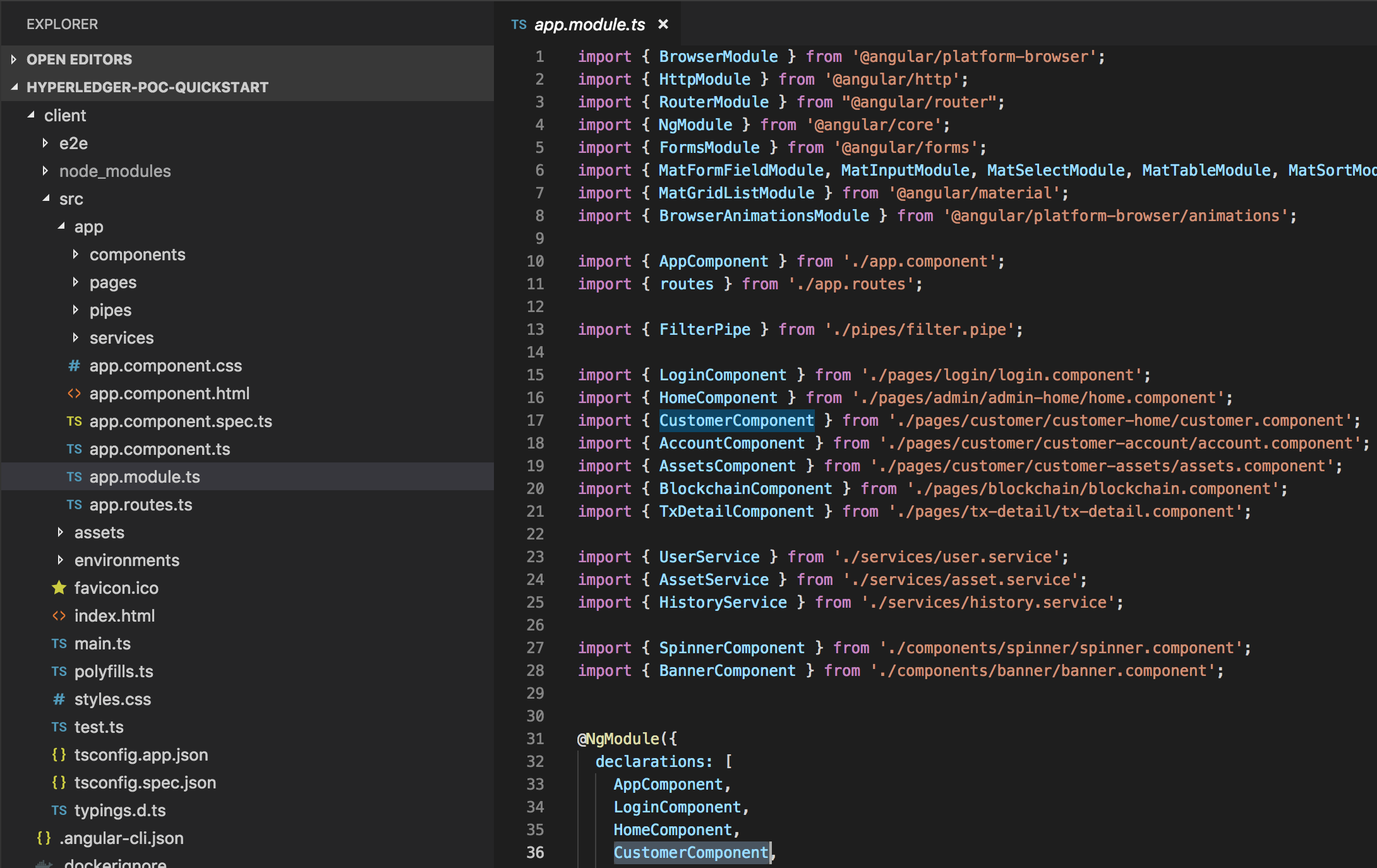
by changing the background color of .navbar-default in the css styles component (client/src/app/components/banner/banner.component.css) and .banner in the main css styles file (client/src/styles.css).

3.2.3 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.

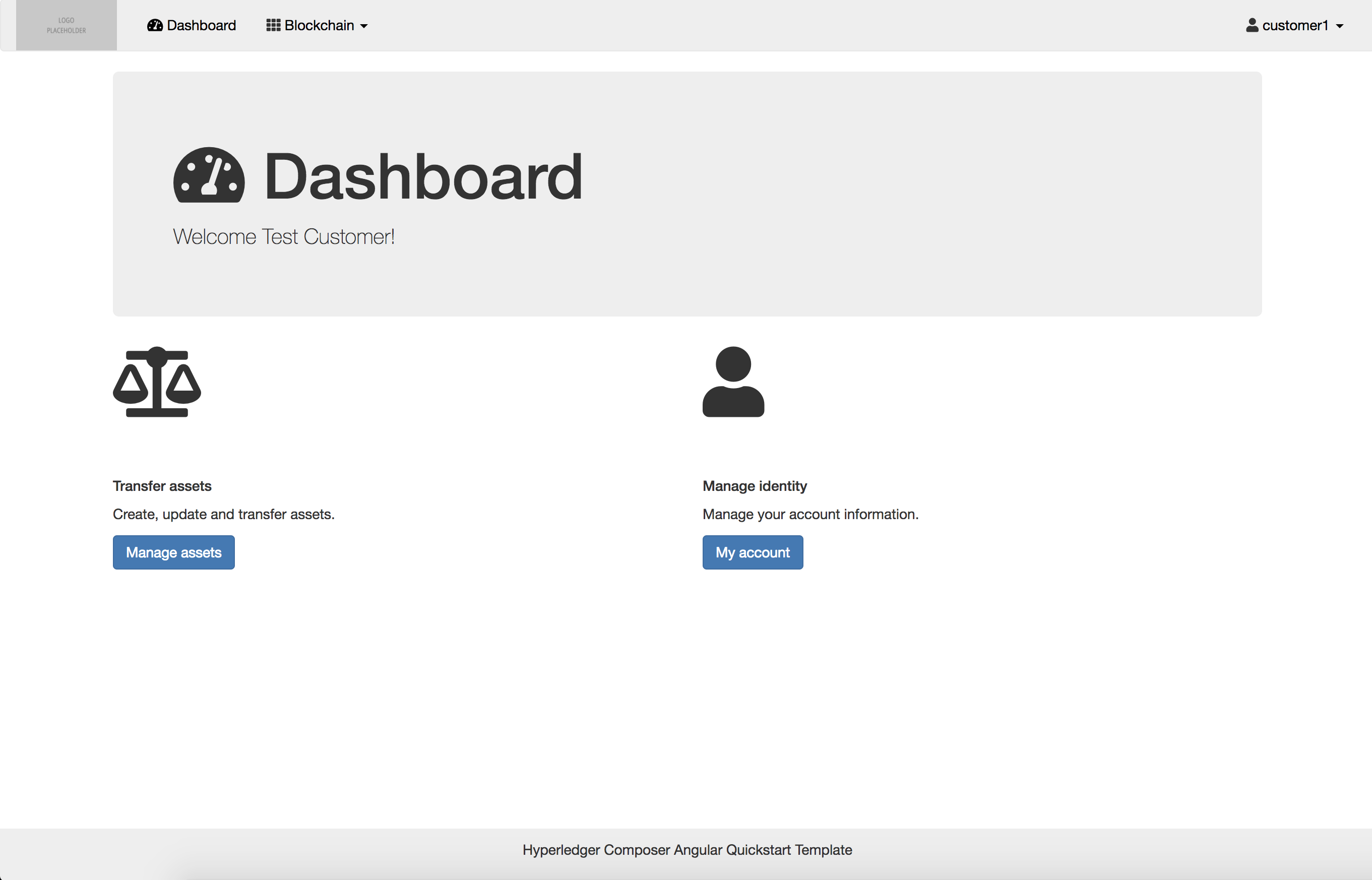
****

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

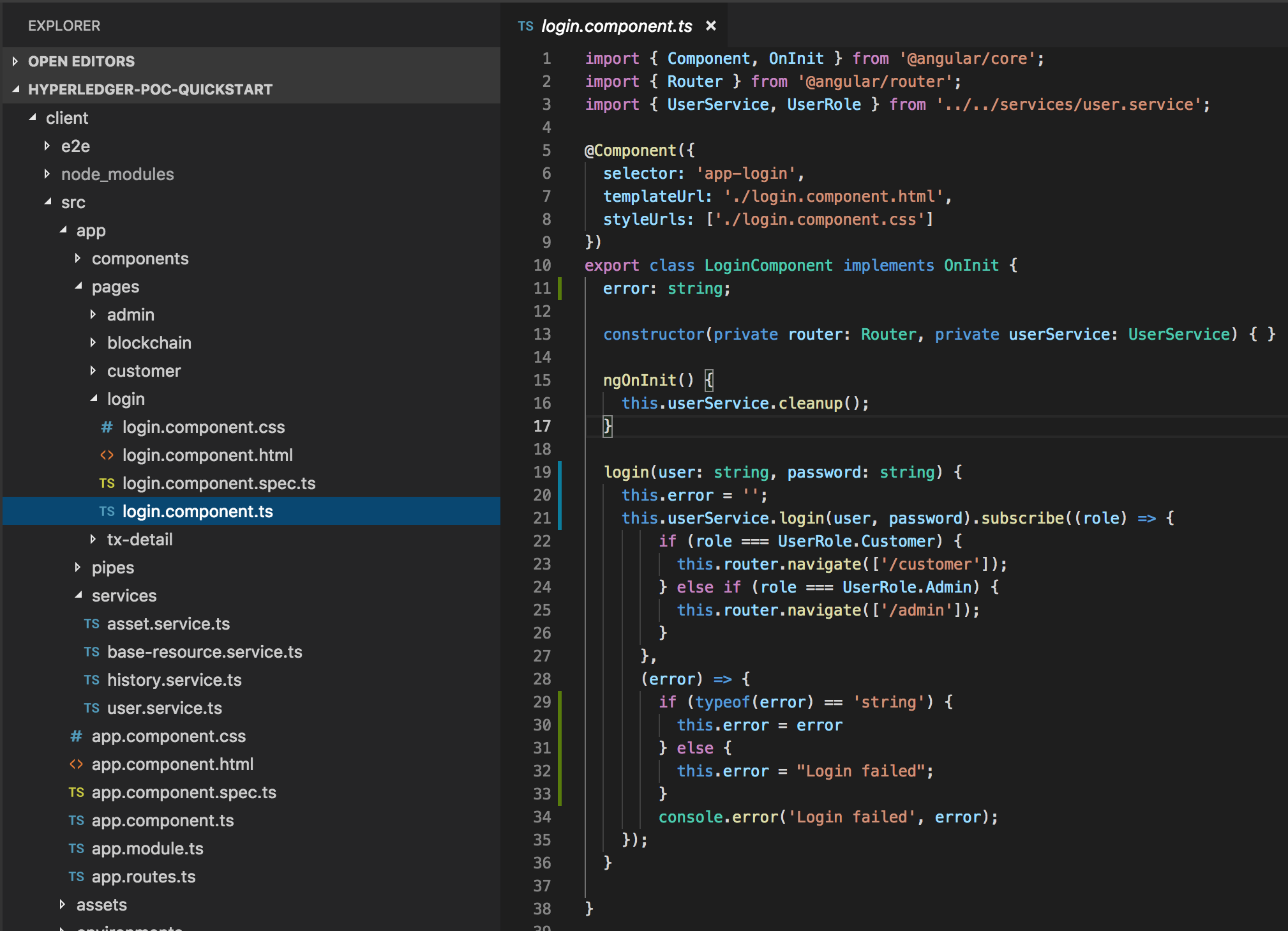




In the demo application, we added a dashboard as homepage for the customer role by creating the Customer component (client/src/app/pages/customer/customer-home/customer.component.ts) and the html content for the page (client/src/app/pages/customer/customer-home/customer.component.html).

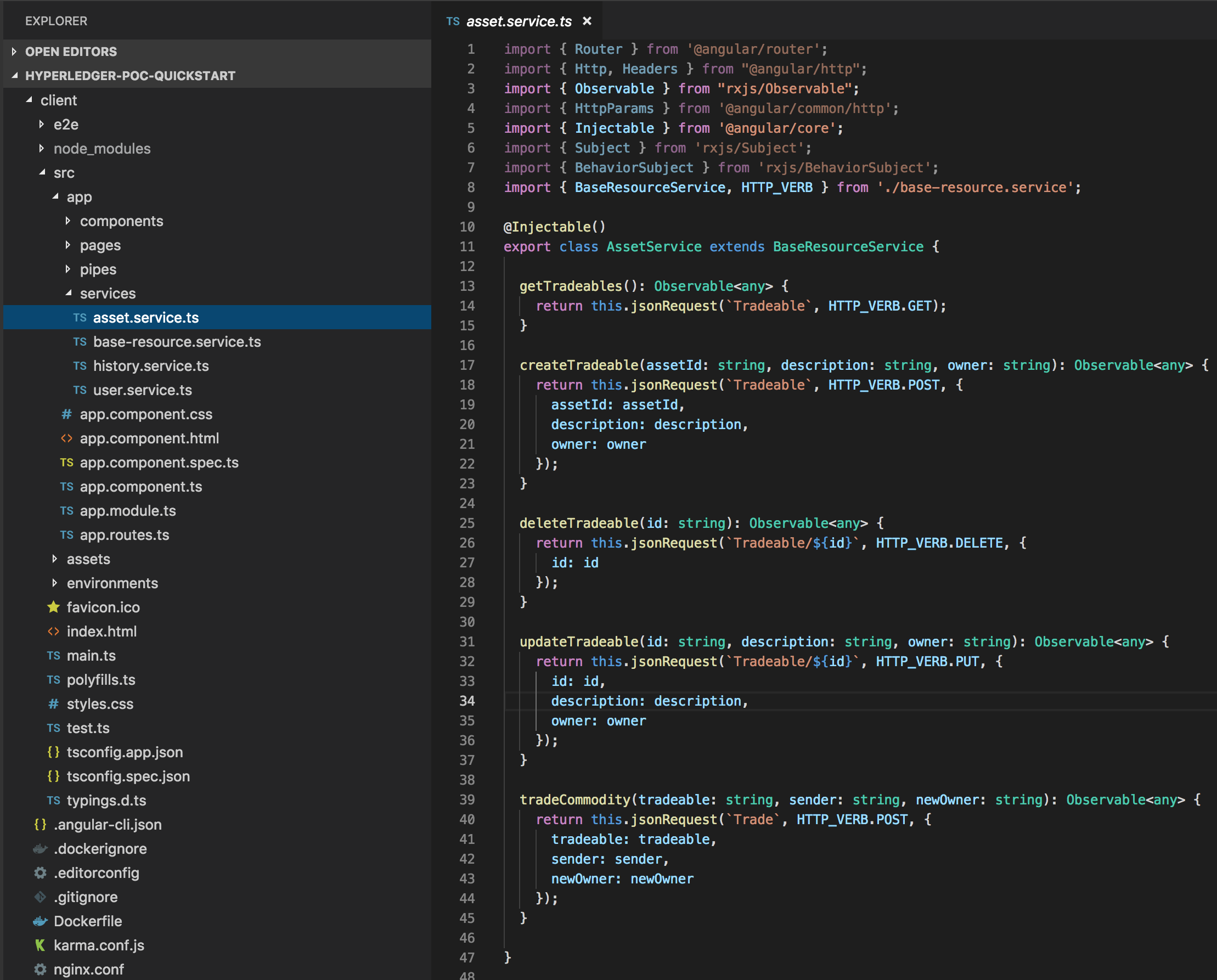
****

3.2.5 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

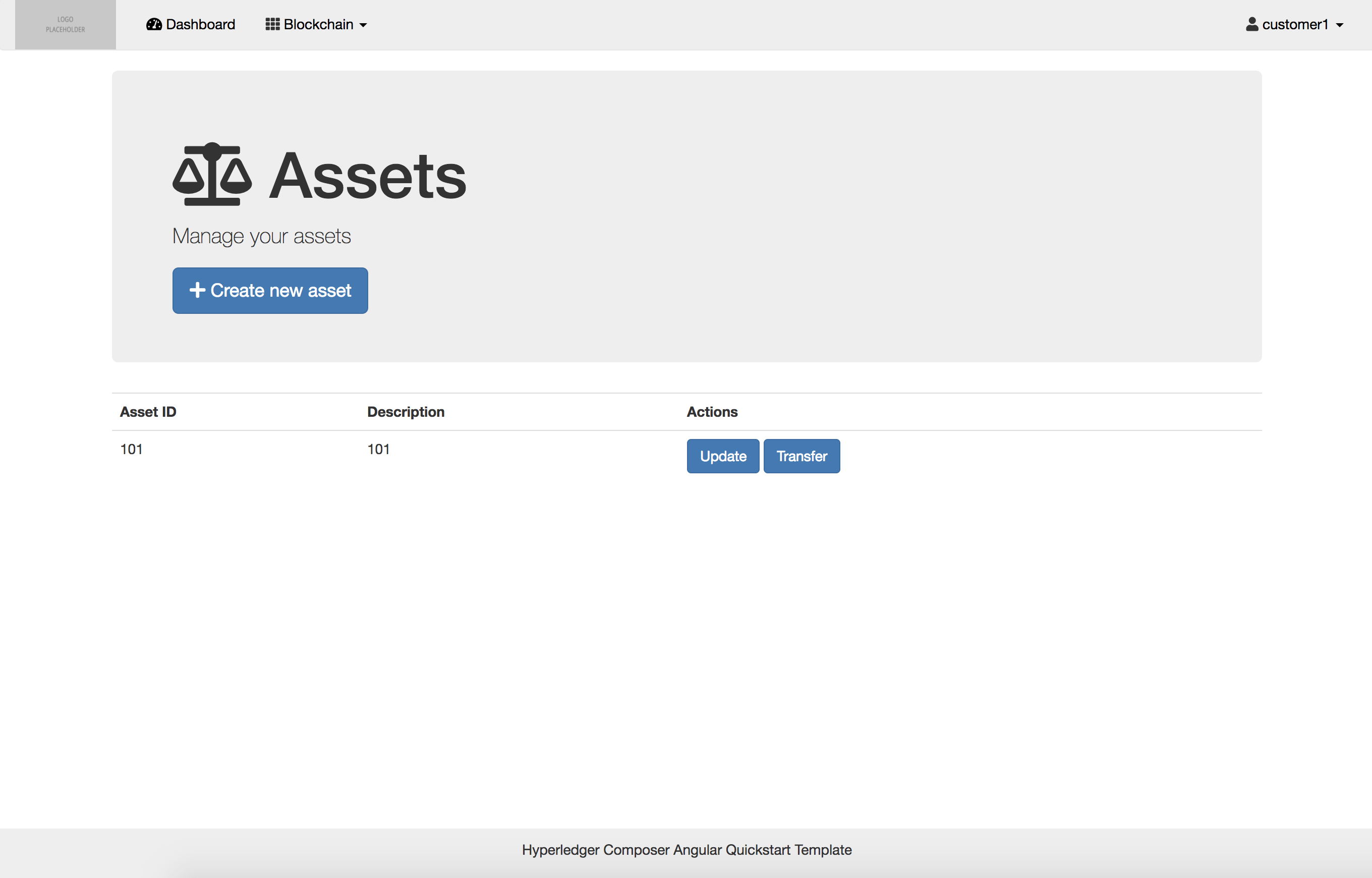


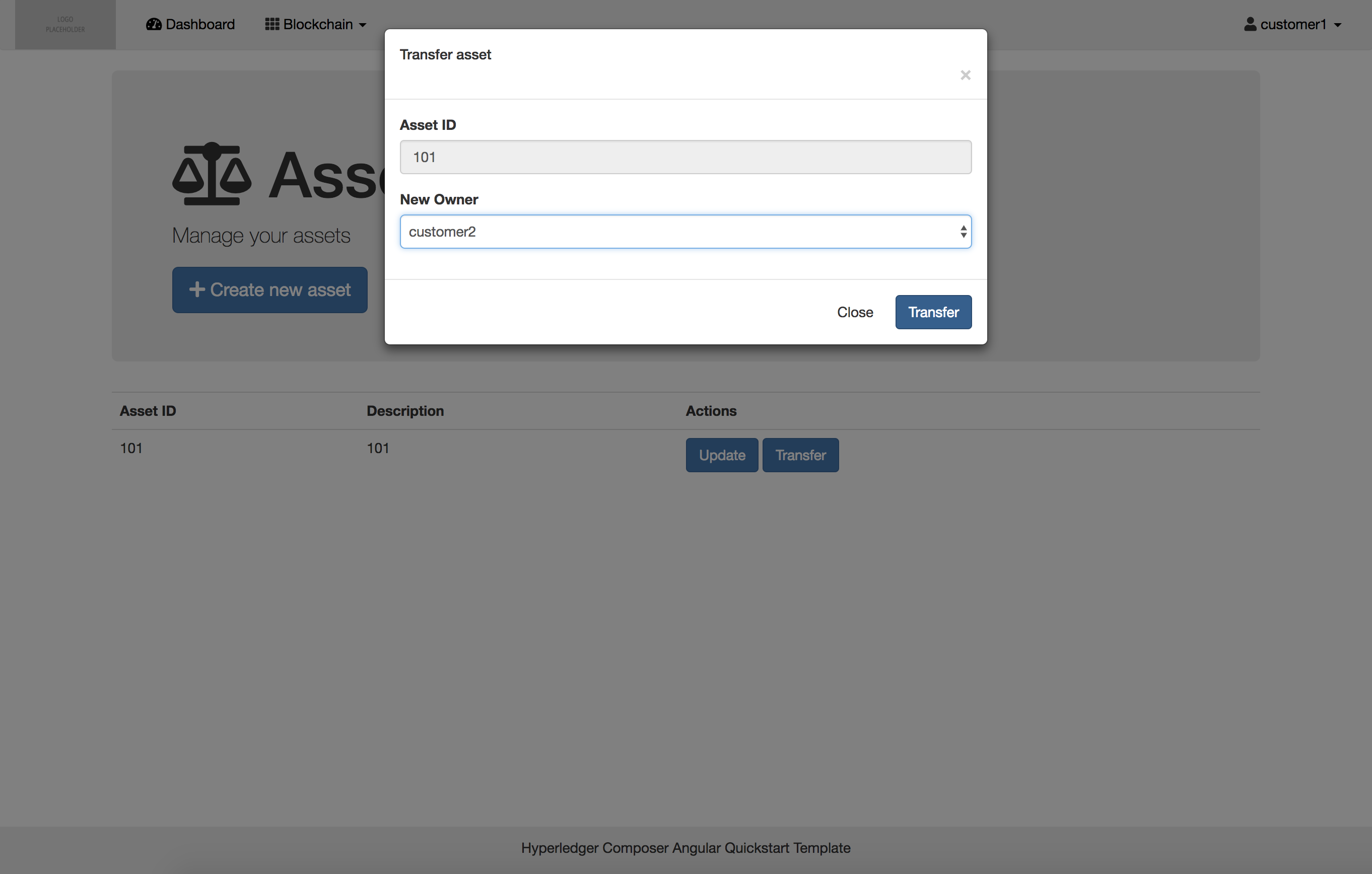
A sample asset transferring demo application is part of the template, which shows you how we have implemented an asset Tradeable and a transaction Trade for the user role Customer. In file asset.service.ts the GET, POST, PUT and DELETE actions are created for the sample asset Tradeable. The tradeCommodity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

3.2.6 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.

****

In the demo, 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.

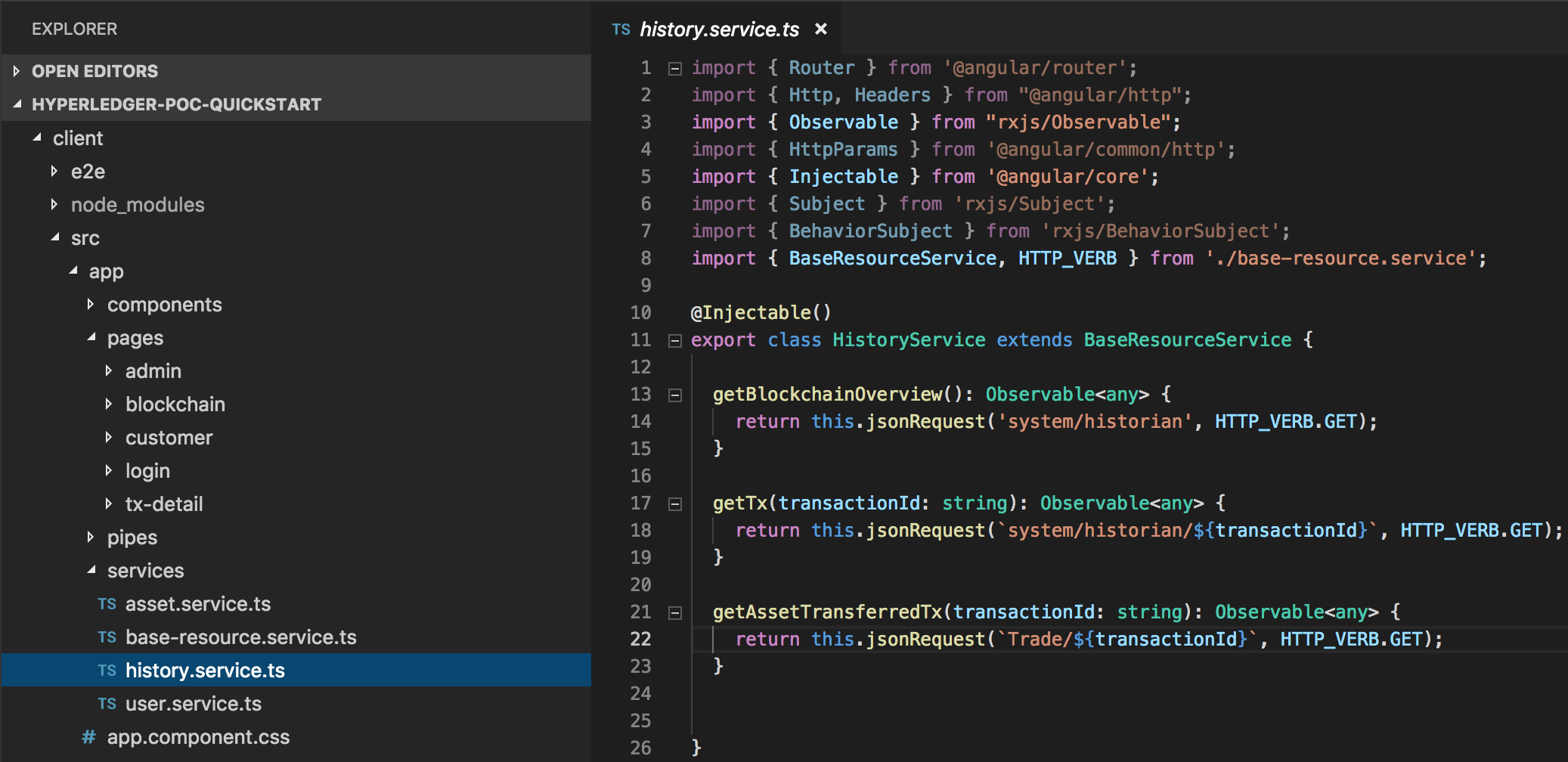
****

****

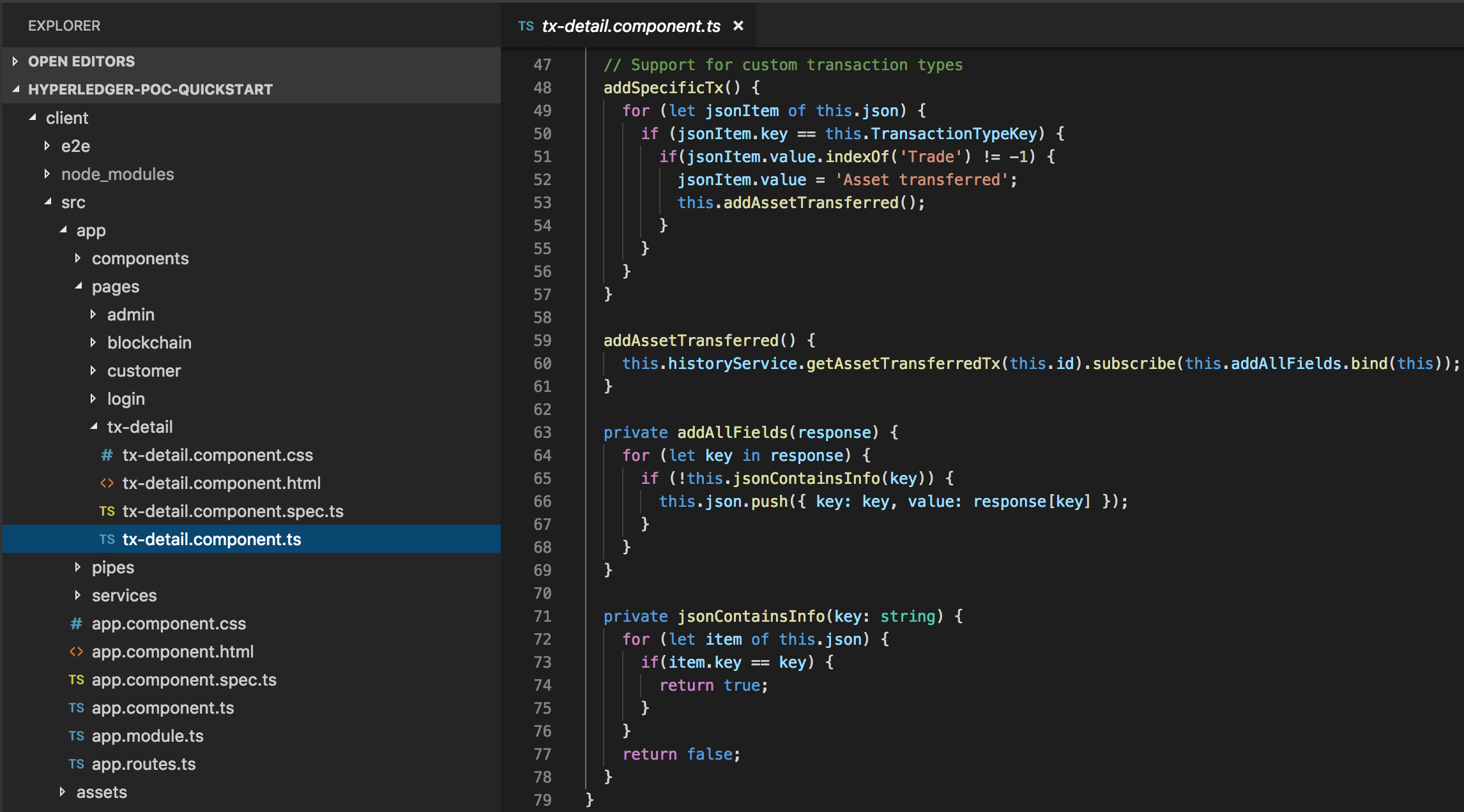
3.2.7 Add a sample asset and transaction   
When you have created the asset.service.ts file in the previous step, then you can use the service to implement the GET, POST, PUT and DELETE actions for assets and the POST action for transactions in the frontend application.

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.

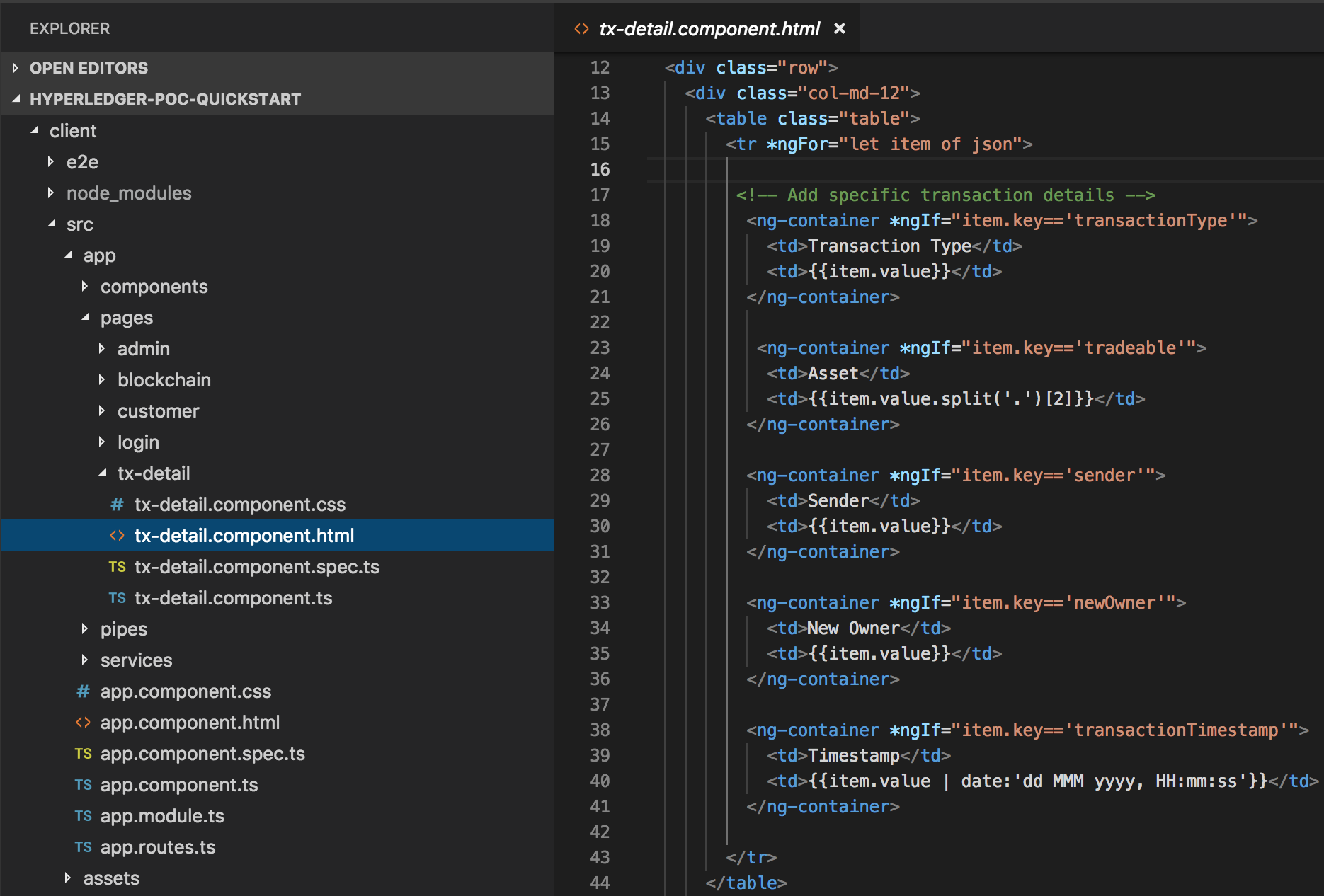
In the demo application, the Trade transaction is used to show how this works. First in the file history.service.ts the getAssetTransferredTx function is created to retrieve all Trade transactions.



In the file 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.



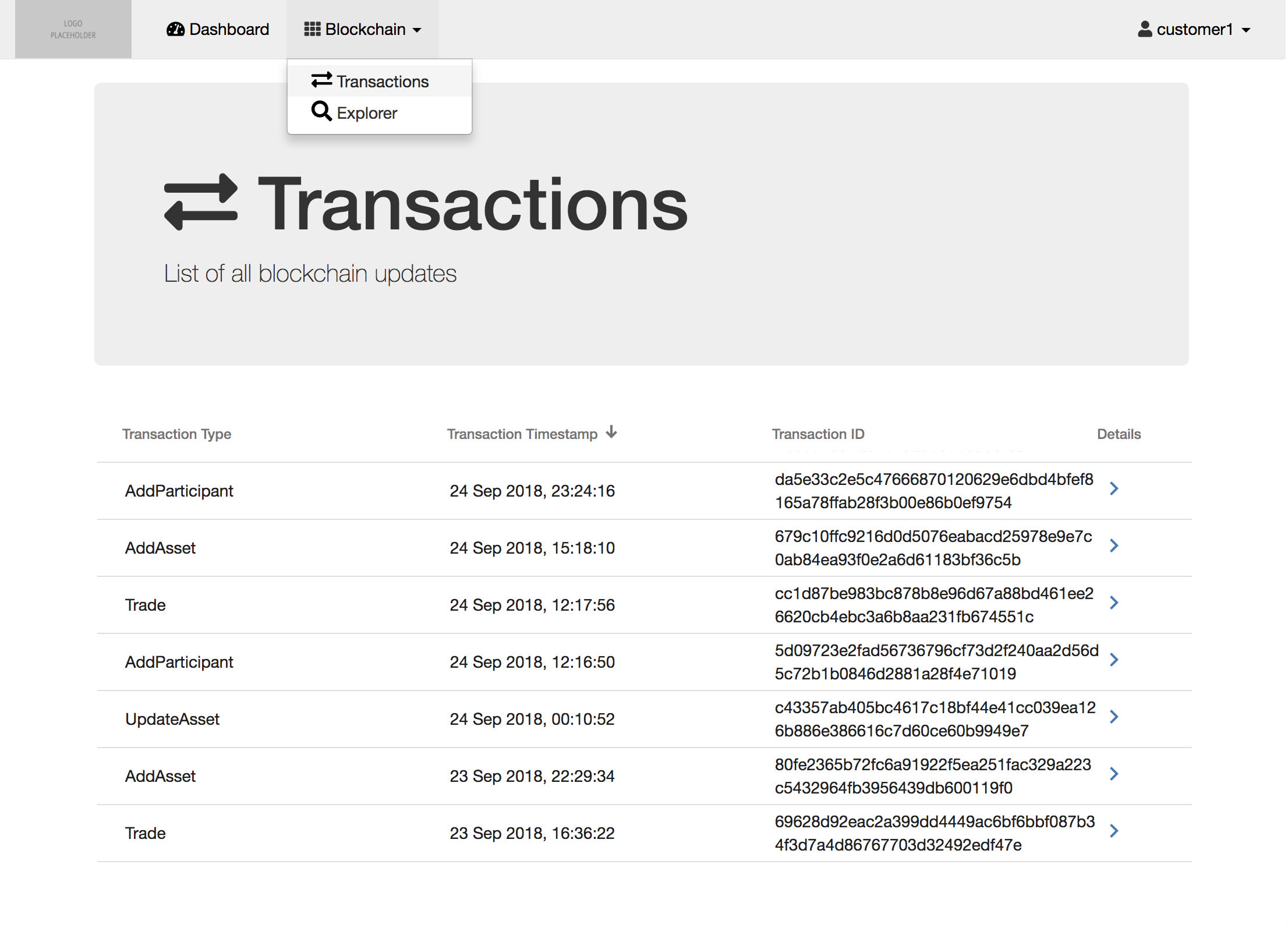
In the tx-detail.component.html file you can update the view of the component. Here you can see how the details of the Trade transactions are shown on the transaction detail page. In this example, we show the transaction type, asset type, sender, new owner and the timestamp of the transaction. Add your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

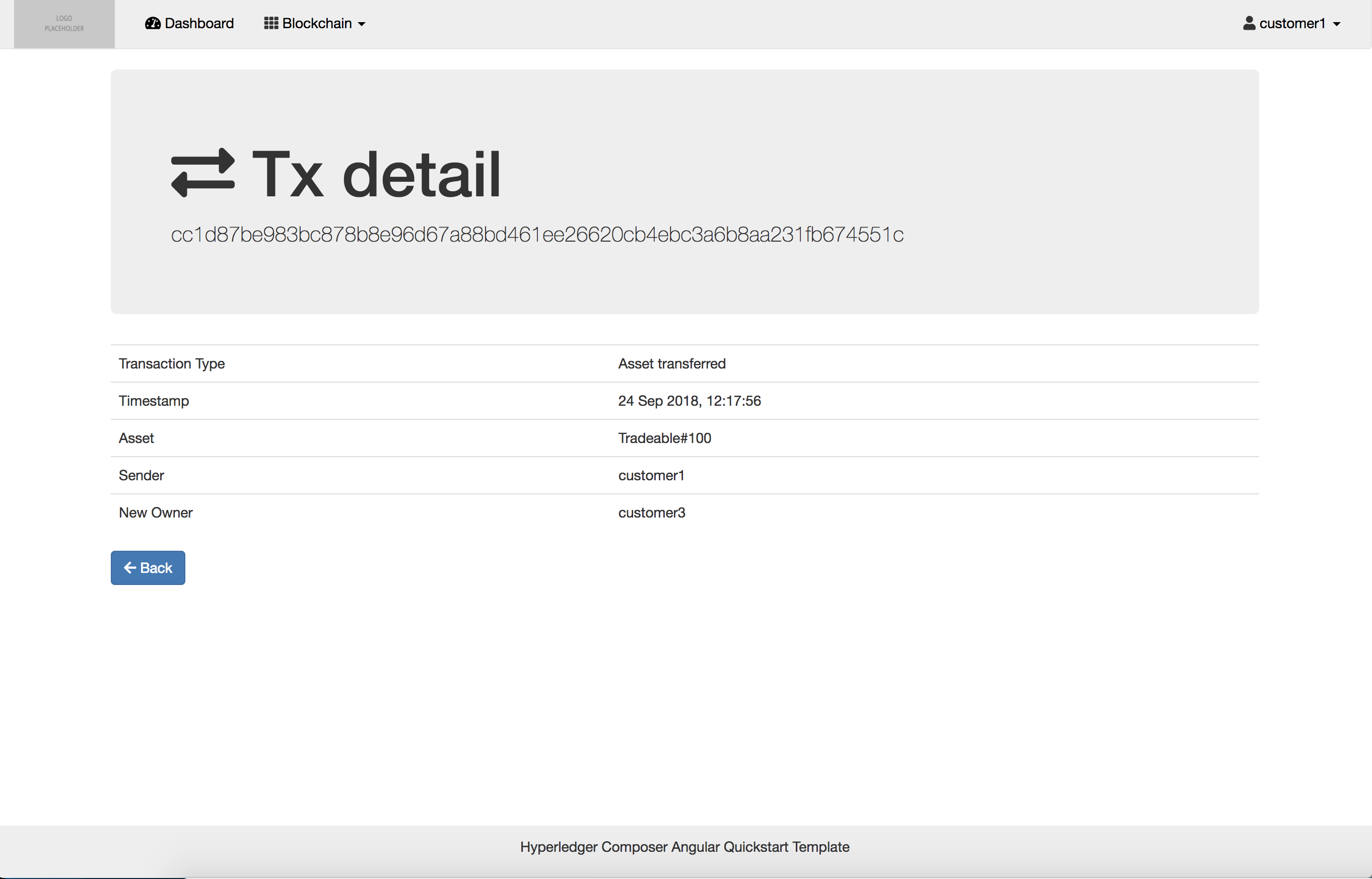


The result is the tx-detail page where the specific transaction details are shown for each individual Trade transaction in the ledger. Go to the top menu in your application, click on blockchain and choose the sub item Transactions. In this list, you see all ledger updates from the start of your successful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

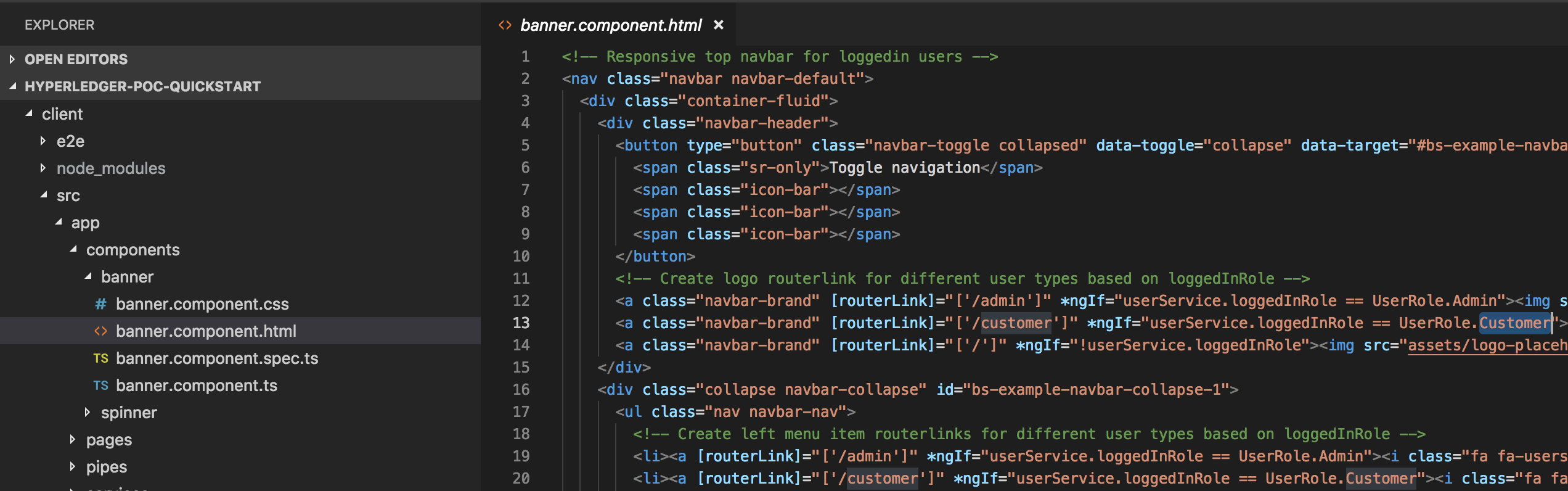
3.2.8 Implement custom transaction monitoring

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.





3.2.9 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).

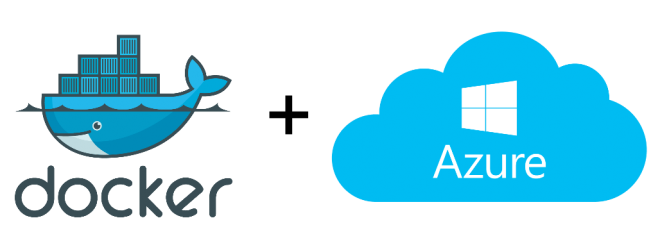


# Deploy app on Azure Cloud

If you want to make your application publically available, deploying it to a cloud-based environment is the easiest and fastest way to go. We have included an automated deployment in this framework to help you to deploy your application to Microsoft’s Azure cloud platform without any hassle.

Please create a free account on Microsoft Azure if you don’t have an account set up already:

<https://azure.microsoft.com/>



## 4.1 Action: Deploy app on Azure Cloud

< Alex his part about the deployment >