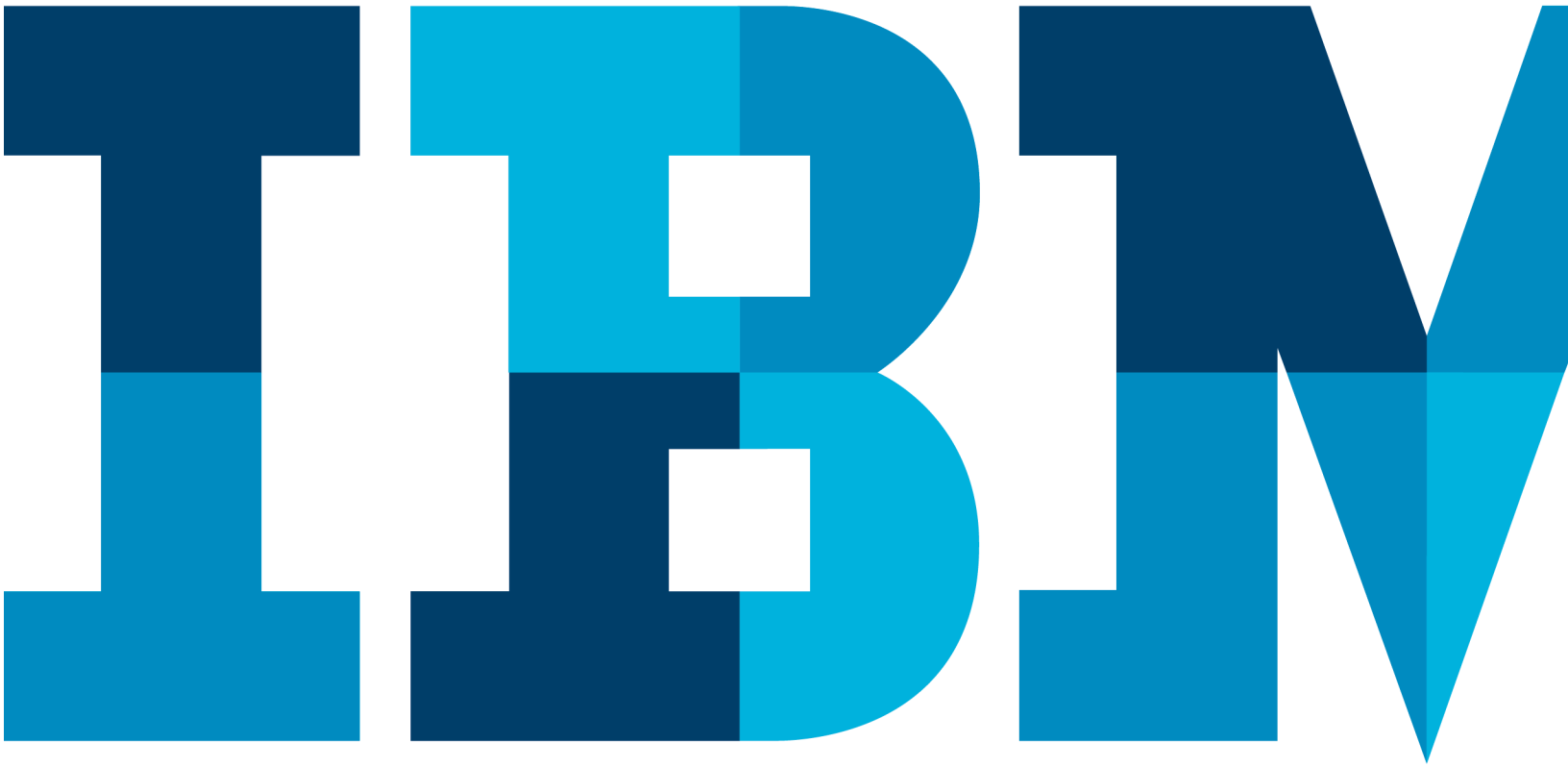
IBM Blockchain Hands-On

IBM Blockchain Platform Visual Studio Code Extension:

Add a Loopback API to your Smart Contract

Lab Seven



IBM

Table of Contents

Disclaimer

IBM's statements regarding its plans, directions, and intent are subject to change or withdrawal without notice at IBM's sole discretion. Information regarding potential future products is intended to outline our general product direction and it should not be relied on in making a purchasing decision.
The information mentioned regarding potential future products is not a commitment, promise, or legal obligation to deliver any material, code or functionality. Information about potential future products may not be incorporated into any contract.

The development, release, and timing of any future features or functionality described for our products remains at our sole discretion I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve results like those stated here.
Information in these presentations (including information relating to products that have not yet been announced by IBM) has been reviewed for accuracy as of the date of initial publication and could include unintentional technical or typographical errors. IBM shall have no responsibility to update this information. **This document is distributed "as is" without any warranty, either express or implied. In no event, shall IBM be liable for any damage arising from the use of this information, including but not limited to, loss of data, business interruption, loss of profit or loss of opportunity.** IBM products and services are warranted per the terms and conditions of the agreements under which they are provided.
IBM products are manufactured from new parts or new and used parts.
In some cases, a product may not be new and may have been previously installed. Regardless, our warranty terms apply."
Any statements regarding IBM's future direction, intent or product plans are subject to change or withdrawal without notice.
Performance data contained herein was generally obtained in controlled, isolated environments. Customer examples are presented as illustrations of how those customers have used IBM products and the results they may have achieved. Actual performance, cost, savings or other results in other operating environments may vary.
References in this document to IBM products, programs, or services does not imply that IBM intends to make such products, programs or services available in all countries in which IBM operates or does business.
Workshops, sessions and associated materials may have been prepared by independent session speakers, and do not necessarily reflect the views of IBM. All materials and discussions are provided for informational purposes only, and are neither intended to, nor shall constitute legal or other guidance or advice to any individual participant or their specific situation.
It is the customer's responsibility to insure its own compliance with legal requirements and to obtain advice of competent legal counsel as to the identification and interpretation of any relevant laws and regulatory requirements that may affect the customer's business and any actions the customer may need to take to comply with such laws. IBM does not provide legal advice or represent or warrant that its services or products will ensure that the customer follows any law.
Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products about this publication and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of

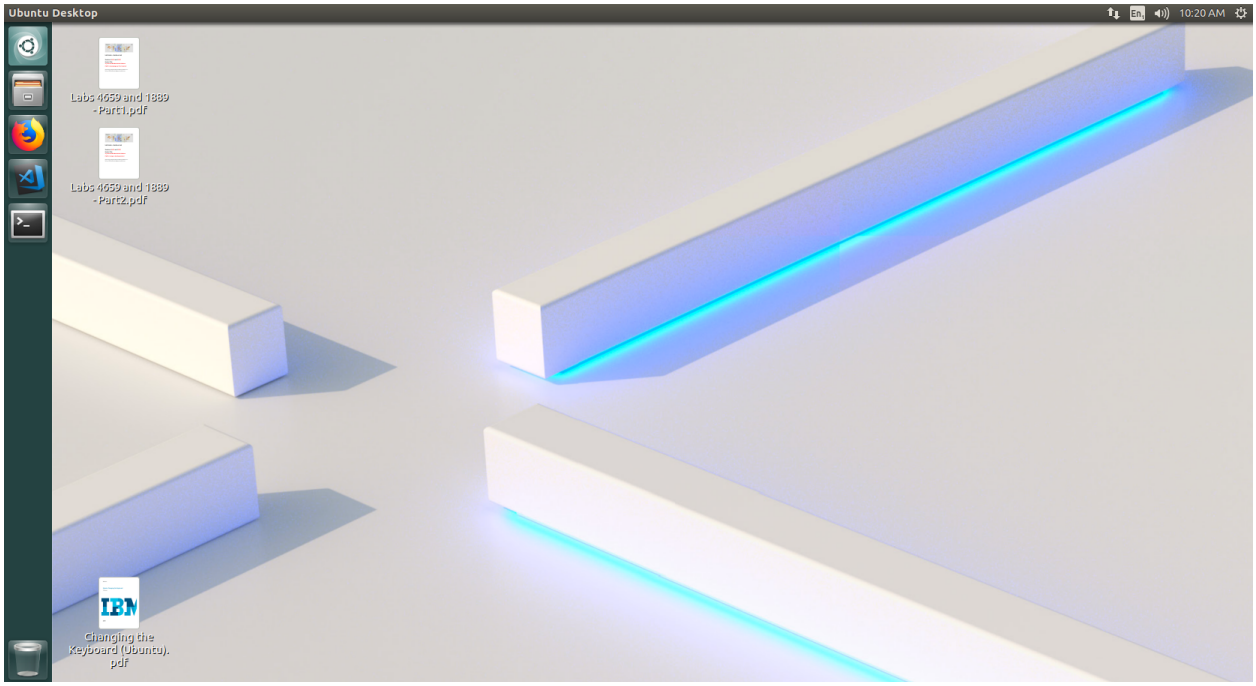
1 Overview of the lab 7 environment and scenario

This lab is a technical introduction to blockchain, specifically smart contract development using the latest developer enhancements in the Linux Foundation's Hyperledger Fabric v1.4 and shows you how IBM's Blockchain Platform's developer experience can accelerate your pace of development.
***Note**: The screenshots in this lab guide were taken using version **1.31.1** of **VSCode**, and version **0.3.0** of the **IBM Blockchain Platform** plugin. If you use different versions, you may see differences those shown in this guide.*
Start here. Instructions are always shown on numbered lines like this one:

__ 1. If it is not already running, start the virtual machine for the lab. The instructor will tell you how to do this if you are unsure.

__ 2. Wait for the image to boot and for the associated services to start. This happens automatically but might take several minutes. The image is ready to use when the desktop is visible as per the screenshot below.

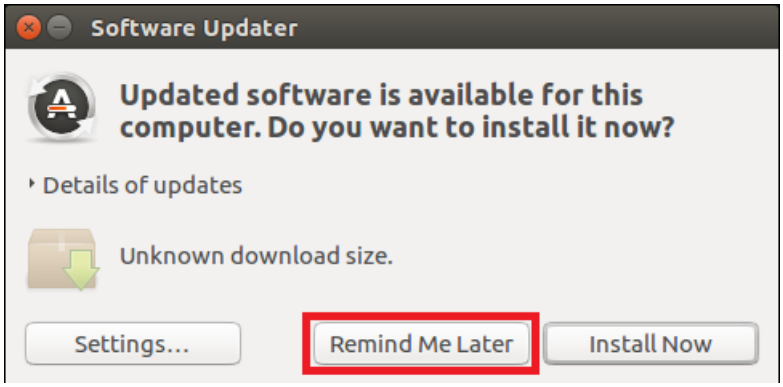
If it asks you to login, the userid and password are both "blockchain".



1.1 Lab 6 Scenario

In this lab, we will examine how loopback can be used to expose chaincode transaction methods as RESTful APIs.

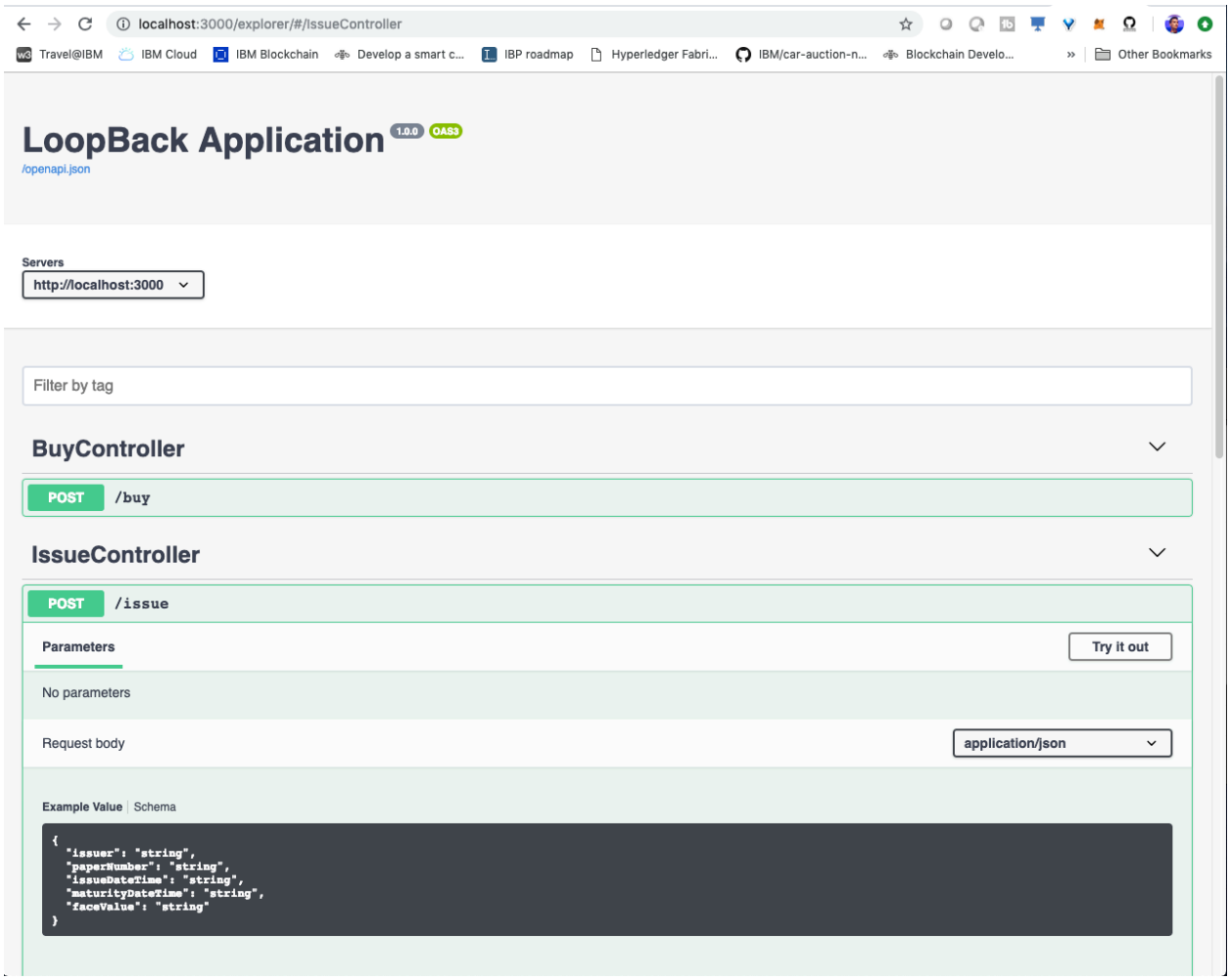
Note *that if you get an "Software Updater" pop-up at any point during the lab, please click "****Remind Me Later****":*



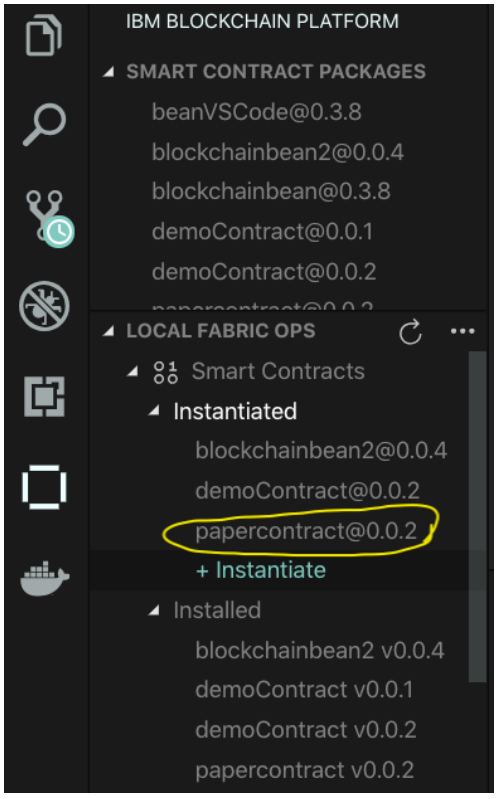
Article I. Commercial Paper Loopback

This is an application that acts as a front end for interacting with the [commercial paper smart contract tutorial] from the Hyperledger Fabric 1.4 documentation. Here is the [video tutorial] that accompanies this repo.

The finished application looks like this:



Note, to use this app, you must have the commercial paper contract deployed as **papercontract** on your Hyperledger Fabric network (Lab 6). I.e. If you are using the [IBM Blockchain VSCode extension], your instantiated contract list should look something like this (i.e. must have **papercontract**) instantiated:

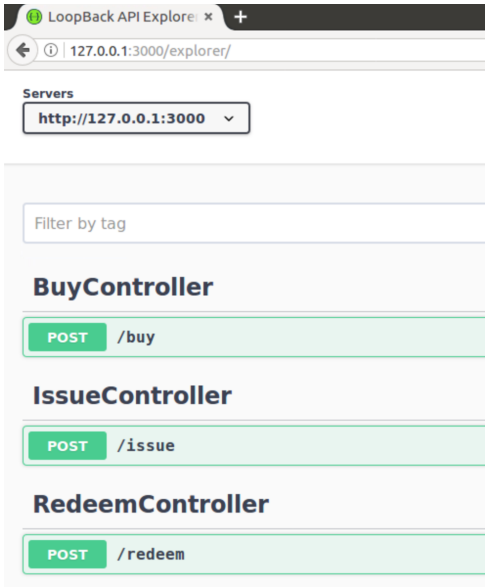


(a) Steps

1. From the left pane toolbar open up a new terminal (Right click for new terminal option).
2. From the newly opened terminal, `cd ~/workspace`
3. From the terminal run the command:

```
git clone https://github.com/horeaporutiu/commercialPaperLoopback.git
```

4. You should now have a newly created commercialPaperLoopback directory under ~/workspace.
5. cd to the new directory. Run `npm install` in the newly cloned directory.
6. run `npm start` in the newly cloned directory
7. From the left pane toolbar, open Firefox. Go to http://127.0.0.1:3000/explorer/ and interact with the contract. You should now see the buy, issue and redeem endpoints.

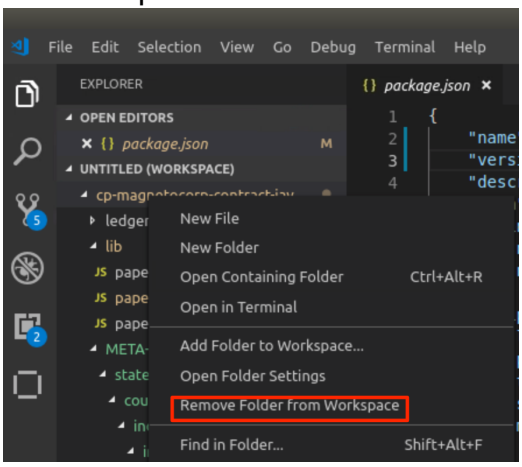


Section 1.02 Overview

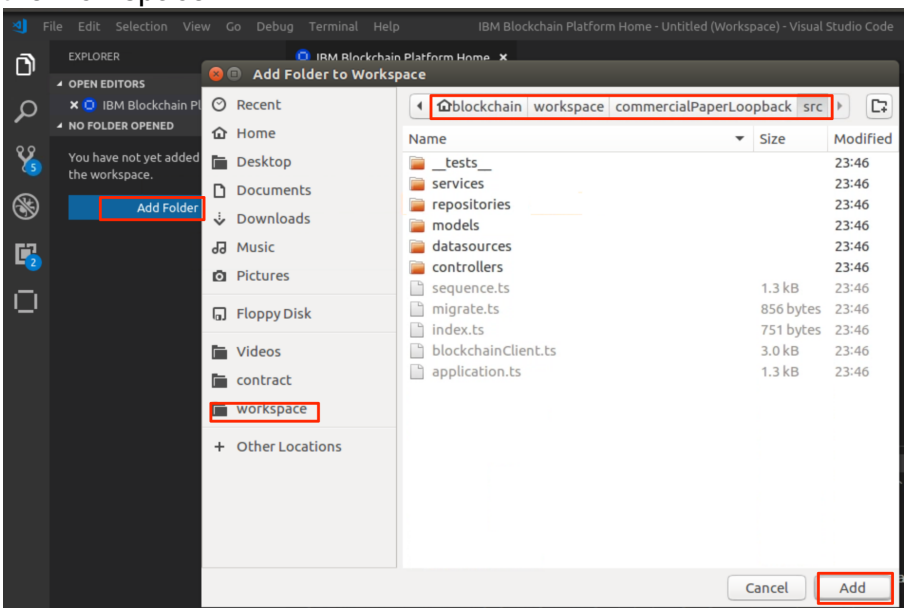
This application shows how to easily scaffold and get started creating a front-end application to interact with your deployed smart contract. This application is an extension to the Hyperledger paper contract developer tutorial, and adds on an extra layer (the UI) with the goal of enabling users that are not developers to interact with a smart contract.

The application has three main components - the models, the views, and the controllers. Since loopback (the lb4 tool) creates the UI for you, based on your models and controllers, I will focus on just those two components (the models and the controllers).

__1. If VSCode isn't already open, Open VSCode using the left hand toolbar. From the explorer view, if there are any open folders, remove any open folders from the workspace



__2. Click **Add Folder** and add the ~/workspace/commercialPaperLoopback folder to the workspace.



(i) Models

To view the models of this application, in VSCode, go to `commercialPaperLoopback/src/models`. Let's first take a look at the `models/issue.model.ts` file. When you open the file, you will see the following code:

```
import {Entity, model, property} from '@loopback/repository';

@model()
export class Issue extends Entity {
```

```
  @property({
    type: 'string',
    required: true,
  })
  issuer: string;

  @property({
    type: 'string',
    required: true,
  })
  paperNumber: string;

  @property({
    type: 'string',
    required: true,
  })
  issueDateTime: string;

  @property({
    type: 'string',
    required: true,
  })
  maturityDateTime?: string;

  @property({
    type: 'string',
  })
  faceValue?: string; // address,city,zipcode

  constructor(data?: Partial<Issue>) {
    super(data);
  }
}
```

What you see here are the definitions of the properties of the Issue object. We can make the properties required - as you can see in the first two properties, or optional. Ok - time to move on to the controllers. Close the file.

(ii) Controllers

The controllers will take what we write into our request body (i.e. the UI of the loopback app) and pass that data to the `papernet` smart contract we have deployed on our local Hyperledger Fabric network. To get an idea of the controllers, from VSCode go to your `commercialPaperLoopback/src/controllers` directory and open the `issue.controller.ts` file. In there, you will see the following code:

```
import { BlockChainModule } from '../blockchainClient';

let blockchainClient = new BlockChainModule.BlockchainClient();

export class IssueController {
```

```
  constructor() {}

  @post('/issue', {
    responses: {
      '200': {
        description: 'Todo model instance',
        content: {'application/json': {schema: {'x-ts-type': Issue}}},
      },
    },
  })
  async createIssue(@requestBody() requestBody: Issue): Promise<Issue> {
    console.log('Buy, requestBody: ')
    console.log(requestBody)


    let networkObj = await blockchainClient.connectToNetwork();
    if (!networkObj) {
      let errString = 'Error connecting to network';
      let issue = new Issue({issuer: errString, paperNumber: errString,
issueDateTime: errString, maturityDateTime: errString });
      return issue;
    }
    console.log('network obj: ')
    console.log(networkObj)

    let dataForIssue = {
      function: 'issue',
      issuer: requestBody.issuer,
      paperNumber: requestBody.paperNumber,
      issueDateTime: requestBody.issueDateTime,
      maturityDateTime: requestBody.maturityDateTime,
      faceValue: requestBody.faceValue,
      contract: networkObj.contract
    };

    var resultAsBuffer = await blockchainClient.issue(dataForIssue);

    console.log('result from blockchainClient.submitTransaction in controller: ')
    let result = JSON.parse(Buffer.from(JSON.parse(resultAsBuffer)).toString())
    let issue = new Issue({issuer: result.issuer, paperNumber: result.paperNumber,
issueDateTime: result.issueDateTime,
      maturityDateTime: result.maturityDateTime
    });
    return issue;
  }

}
```

At the top of the file, we define our response schema - the data structure that will be returned to the UI after submitting a transaction to the smart contract. Notice that here we are returning an object of type `Issue` - which we have defined in our `issue.model.ts` file.

Next, we define the CRUD operations of our API. For simplicity - we have only defined the `/post/issue` method. Note that all CRUD methods can be implemented in a similar manner to `/post/issue`.

In our `async createIssue(@requestBody() requestBody: Issue): Promise<Issue> {` method, we first connect to our Hyperledger Fabric network by telling our application where our peers, orderers, and certificate authority are running - this is all done by calling the `blockchainClient.connectToNetwork()` function, which we have imported at the top of the file. (see blockchainClient.ts).

Next, we look for a deployed contract such as the paper net contract. Finally, we call the appropriate method in the deployed contract, and send that response back to the user.

Thanks for taking a quick tour of some sample Loopback code!