/\*

\* Licensed to the Apache Software Foundation (ASF) under one

\* or more contributor license agreements. See the NOTICE file

\* distributed with this work for additional information

\* regarding copyright ownership. The ASF licenses this file

\* to you under the Apache License, Version 2.0 (the

\* "License"); you may not use this file except in compliance

\* with the License. You may obtain a copy of the License at

\*

\* http://www.apache.org/licenses/LICENSE-2.0

\*

\* Unless required by applicable law or agreed to in writing,

\* software distributed under the License is distributed on an

\* "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY

\* KIND, either express or implied. See the License for the

\* specific language governing permissions and limitations

\* under the License.

\*/

/\*

\* The sample smart contract for documentation topic:

\* Writing Your First Blockchain Application

\*/

package main

/\* Imports

\* 4 utility libraries for formatting, handling bytes, reading and writing JSON, and string manipulation

\* 2 specific Hyperledger Fabric specific libraries for Smart Contracts

\*/

import (

"bytes"

"encoding/json"

"fmt"

"strconv"

"github.com/hyperledger/fabric/core/chaincode/shim"

sc "github.com/hyperledger/fabric/protos/peer"

)

// Define the Smart Contract structure

type SmartContract struct {

}

// Define the car structure, with 4 properties. Structure tags are used by encoding/json library

type Rep struct {

Name string `json:"name"`

DocName string `json:"docname"`

Allergy string `json:"allergy"`

Height string `json:"height"`

Weight string `json:"weight"`

BloodPressure string `json:"bloodpressure"`

BloodType string `json:"bloodtype"`

Insurance string `json:"insurance"`

Medication string `json:"medication"`

}

/\*

\* The Init method is called when the Smart Contract "fabcar" is instantiated by the blockchain network

\* Best practice is to have any Ledger initialization in separate function -- see initLedger()

\*/

func (s \*SmartContract) Init(APIstub shim.ChaincodeStubInterface) sc.Response {

return shim.Success(nil)

}

/\*

\* The Invoke method is called as a result of an application request to run the Smart Contract "fabcar"

\* The calling application program has also specified the particular smart contract function to be called, with arguments

\*/

func (s \*SmartContract) Invoke(APIstub shim.ChaincodeStubInterface) sc.Response {

// Retrieve the requested Smart Contract function and arguments

function, args := APIstub.GetFunctionAndParameters()

// Route to the appropriate handler function to interact with the ledger appropriately

if function == "queryRep" {

return s.queryRep(APIstub, args)

} else if function == "initLedger" {

return s.initLedger(APIstub)

} else if function == "createRep" {

return s.createRep(APIstub, args)

} else if function == "queryAllReps" {

return s.queryAllReps(APIstub)

} else if function == "changeDoctor" {

return s.changeDoctor(APIstub, args)

}

return shim.Error("Invalid Smart Contract function name.")

}

func (s \*SmartContract) queryRep(APIstub shim.ChaincodeStubInterface, args []string) sc.Response {

if len(args) != 1 {

return shim.Error("Incorrect number of arguments. Expecting 1")

}

repAsBytes, \_ := APIstub.GetState(args[0])

return shim.Success(repAsBytes)

}

func (s \*SmartContract) initLedger(APIstub shim.ChaincodeStubInterface) sc.Response {

reps := []Rep{

Rep{Name:"Dave" ,DocName:"Hamid", Allergy:"none", Height:"6\'11\"", Weight:"65" , BloodPressure:"150/102" , BloodType:"A+" , Insurance:"yes" , Medication:"Diuretic" },

Rep{Name:"Angie" ,DocName:"Jonathan" , Allergy:"Dust" , Height:"5\'5\"" , Weight:"50" , BloodPressure:"123/80" , BloodType:"A+" , Insurance:"yes" , Medication:"none"},

Rep{Name:"Thomas" ,DocName:"Davis" , Allergy:"none" , Height:"5\'9\"" , Weight:"56" , BloodPressure:"130/80" , BloodType:"B+" , Insurance:"yes" , Medication:"Rauvolfia serpentina 1x" },

Rep{Name:"Joanne" ,DocName:"Erin" , Allergy:"none" , Height:"6\'2\"" , Weight:"68" , BloodPressure:"117/76" , BloodType:"O+" , Insurance:"yes" , Medication:"none" },

Rep{Name:"Graham" ,DocName:"Luna" , Allergy:"peanuts" , Height:"5\'4\"" , Weight:45 , BloodPressure:"115/75" , BloodType:"B+" , Insurance:"no" , Medication:"none" },

}

i := 0

for i < len(reps) {

fmt.Println("i is ", i)

repAsBytes, \_ := json.Marshal(reps[i])

APIstub.PutState("REPORT"+strconv.Itoa(i), repAsBytes)

fmt.Println("Added", reps[i])

i = i + 1

}

return shim.Success(nil)

}

func (s \*SmartContract) createRep(APIstub shim.ChaincodeStubInterface, args []string) sc.Response {

if len(args) != 5 {

return shim.Error("Incorrect number of arguments. Expecting 5")

}

var rep = Rep{Name: args[1], DoctorName: args[2], Allergy: args[3], Height: args[4], Weight: args[5], BloodPressure: args[6], BloodType: args[7], Insurance: args[8], Medication: args[9]}

repAsBytes, \_ := json.Marshal(rep)

APIstub.PutState(args[0], repAsBytes)

return shim.Success(nil)

}

func (s \*SmartContract) queryAllReps(APIstub shim.ChaincodeStubInterface) sc.Response {

startKey := "REP0"

endKey := "REP999"

resultsIterator, err := APIstub.GetStateByRange(startKey, endKey)

if err != nil {

return shim.Error(err.Error())

}

defer resultsIterator.Close()

// buffer is a JSON array containing QueryResults

var buffer bytes.Buffer

buffer.WriteString("[")

bArrayMemberAlreadyWritten := false

for resultsIterator.HasNext() {

queryResponse, err := resultsIterator.Next()

if err != nil {

return shim.Error(err.Error())

}

// Add a comma before array members, suppress it for the first array member

if bArrayMemberAlreadyWritten == true {

buffer.WriteString(",")

}

buffer.WriteString("{\"Key\":")

buffer.WriteString("\"")

buffer.WriteString(queryResponse.Key)

buffer.WriteString("\"")

buffer.WriteString(", \"Record\":")

// Record is a JSON object, so we write as-is

buffer.WriteString(string(queryResponse.Value))

buffer.WriteString("}")

bArrayMemberAlreadyWritten = true

}

buffer.WriteString("]")

fmt.Printf("- queryAllReps:\n%s\n", buffer.String())

return shim.Success(buffer.Bytes())

}

func (s \*SmartContract) changeDoctor(APIstub shim.ChaincodeStubInterface, args []string) sc.Response {

if len(args) != 2 {

return shim.Error("Incorrect number of arguments. Expecting 2")

}

repAsBytes, \_ := APIstub.GetState(args[0])

rep := Rep{}

json.Unmarshal(repAsBytes, &rep)

rep.Owner = args[1]

repAsBytes, \_ = json.Marshal(rep)

APIstub.PutState(args[0], repAsBytes)

return shim.Success(nil)

}

// The main function is only relevant in unit test mode. Only included here for completeness.

func main() {

// Create a new Smart Contract

err := shim.Start(new(SmartContract))

if err != nil {

fmt.Printf("Error creating new Smart Contract: %s", err)

}

}