A Simple Write/Read Example with R3 Corda

Eric Lee
cheng-yu.eric.lee@accenture.com

Objectives

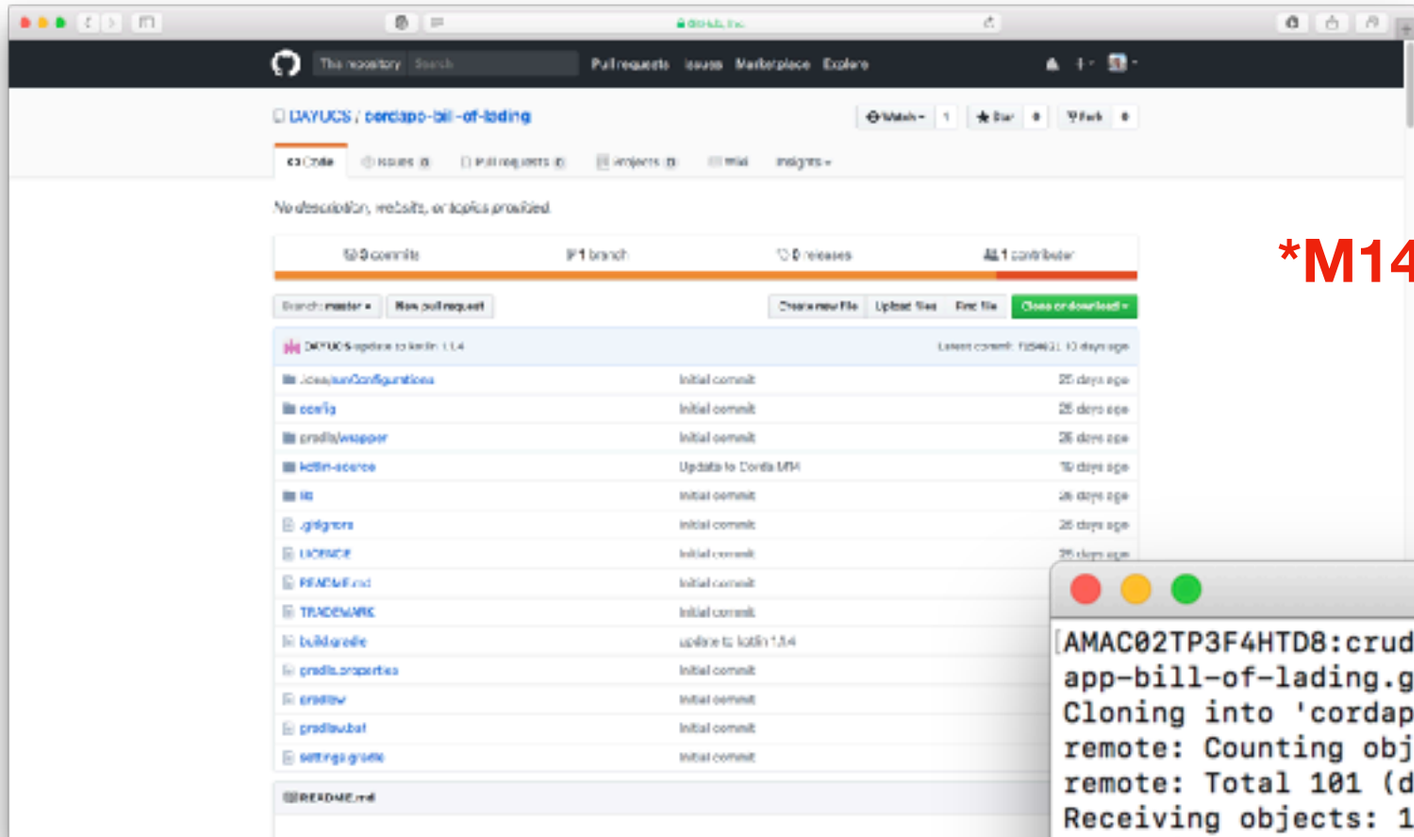
- Define a database schema in Corda “State”
- Define the contract rules and data flows
- Unit test design
- Create REST API to write and read the data
- Test with POSTMAN

Preparation

- Install Oracle Java JDK 8 (Build >139)
- Install IntelliJ 2017.2
- Clone example code from git repository

Clone the Example From Git

```
git clone https://github.com/DAYUCS/cordapp-bill-of-lading.git
```



*M14 based

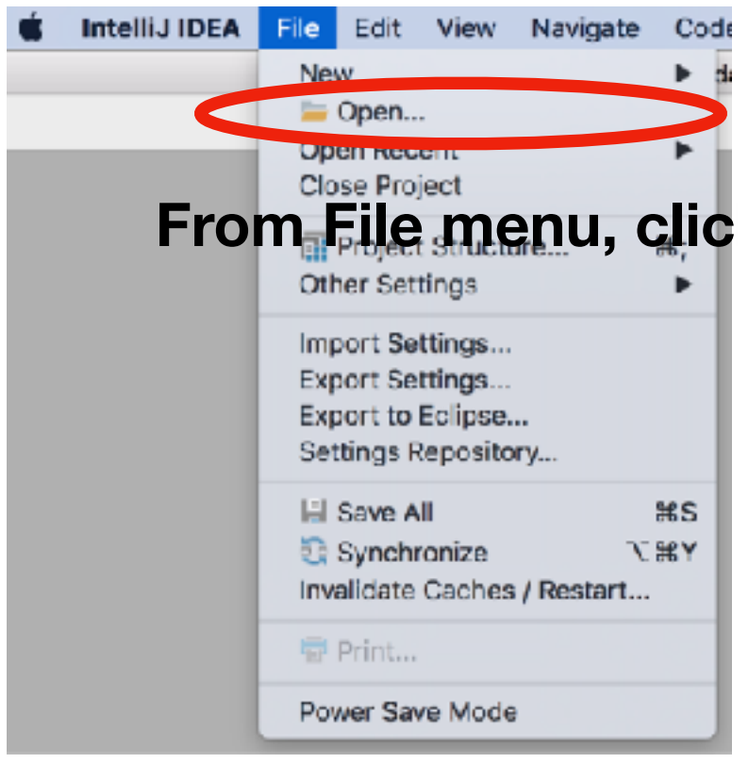
Define the Schema

date	seller	buyer	referenceNumber	totalAmount

I want to record this five columns to ledger as order information.

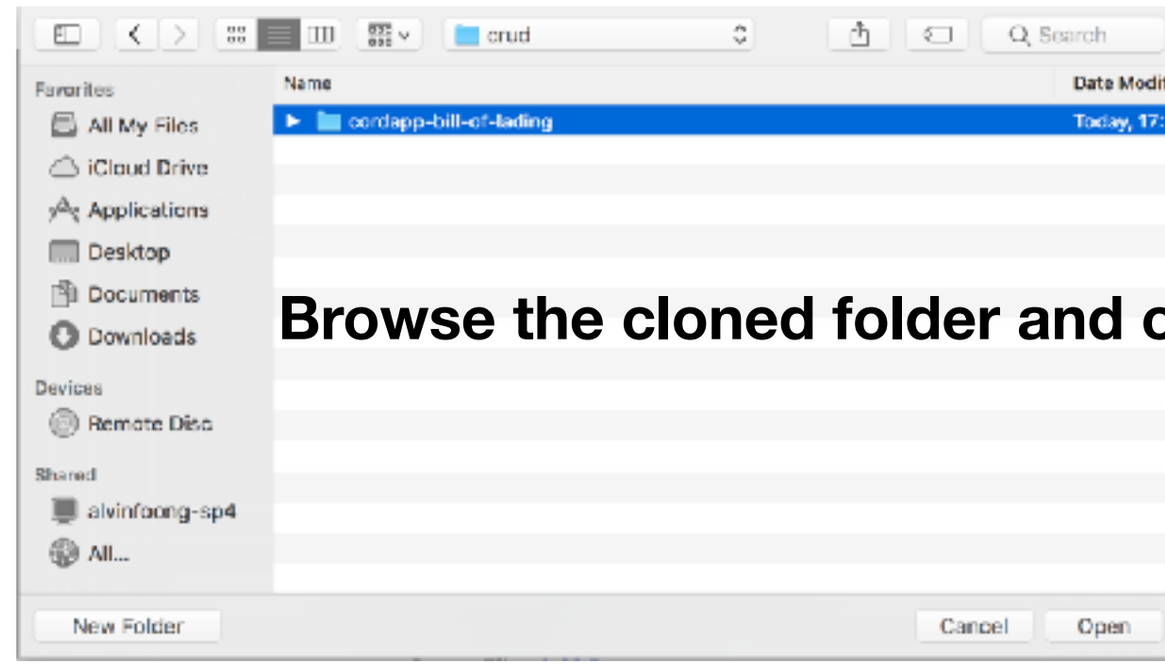
Open the Cloned Project by IntelliJ 2017.02

1



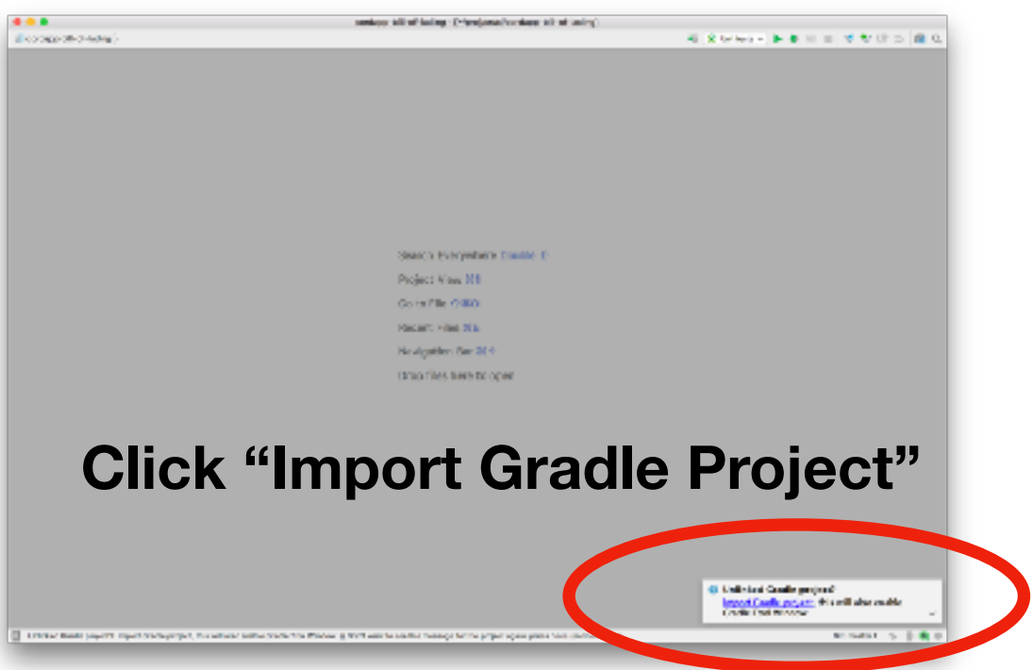
From File menu, click “Open”

2



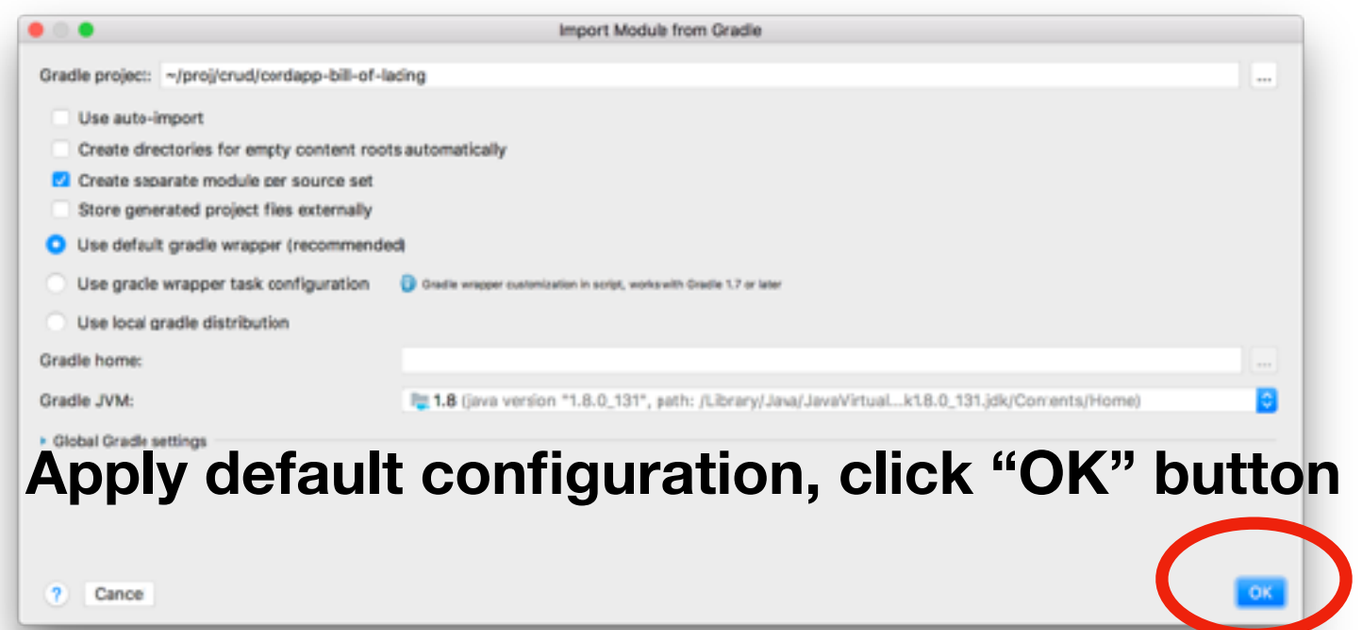
Browse the cloned folder and open

3



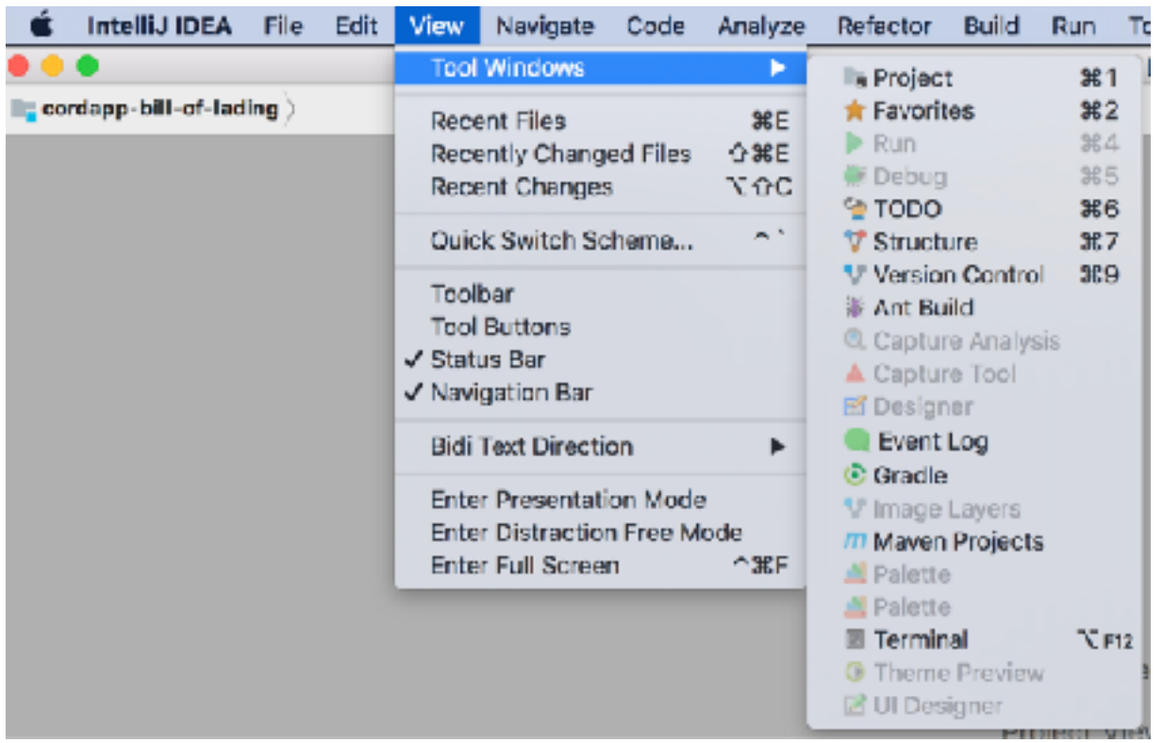
Click “Import Gradle Project”

4



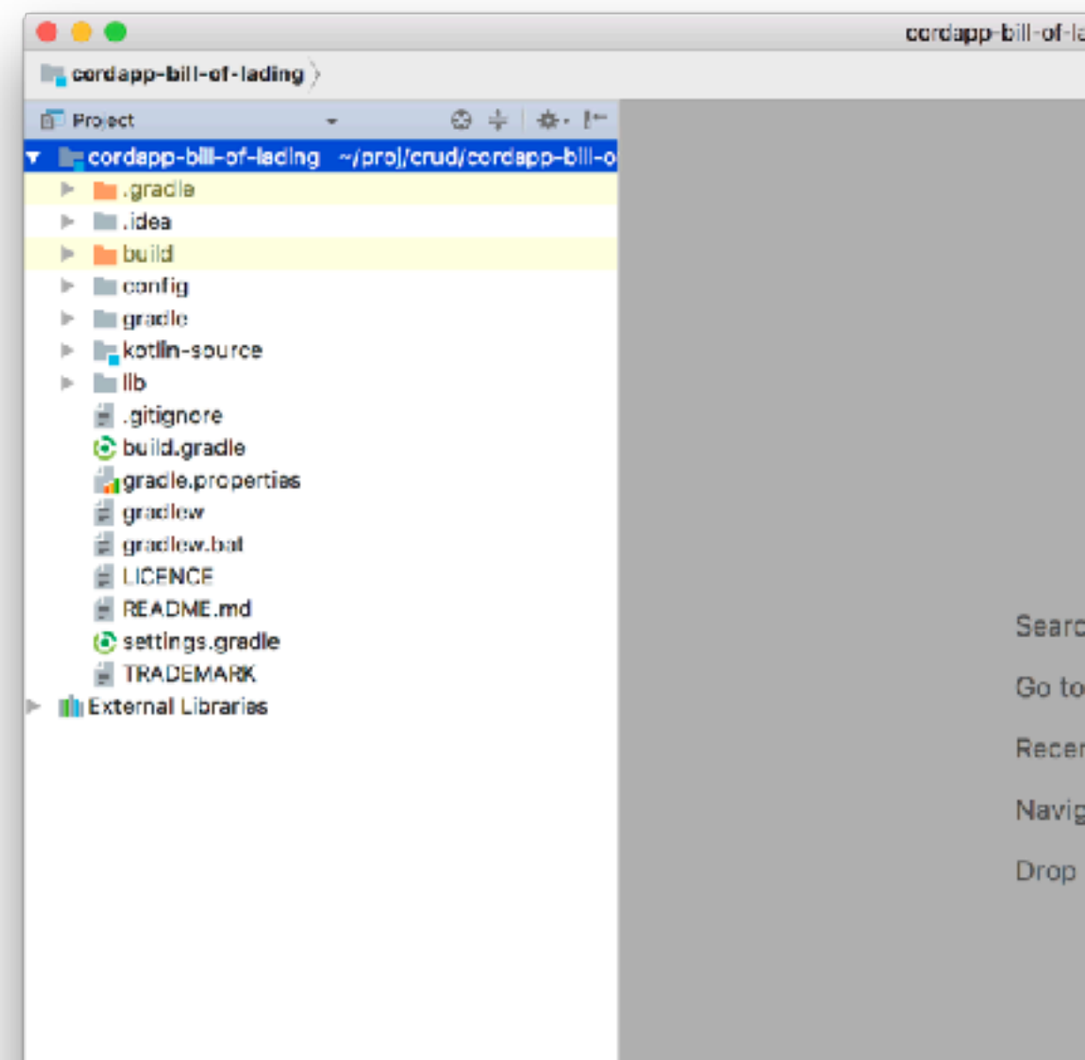
Apply default configuration, click “OK” button

Add IDE Panels



View > Tool Windows > Project

Can see the folder structure now.



You can add “Terminal” panel too.

Modelling the Solution with Corda

- Define the State
  The data model, the record you want to store in the ledger, define the table schema.

- Define the Contract
  The relationships between participants, obligations and the rules.

- Define the Flow
  The workflow for data commitment.

State Related Files (Classes)

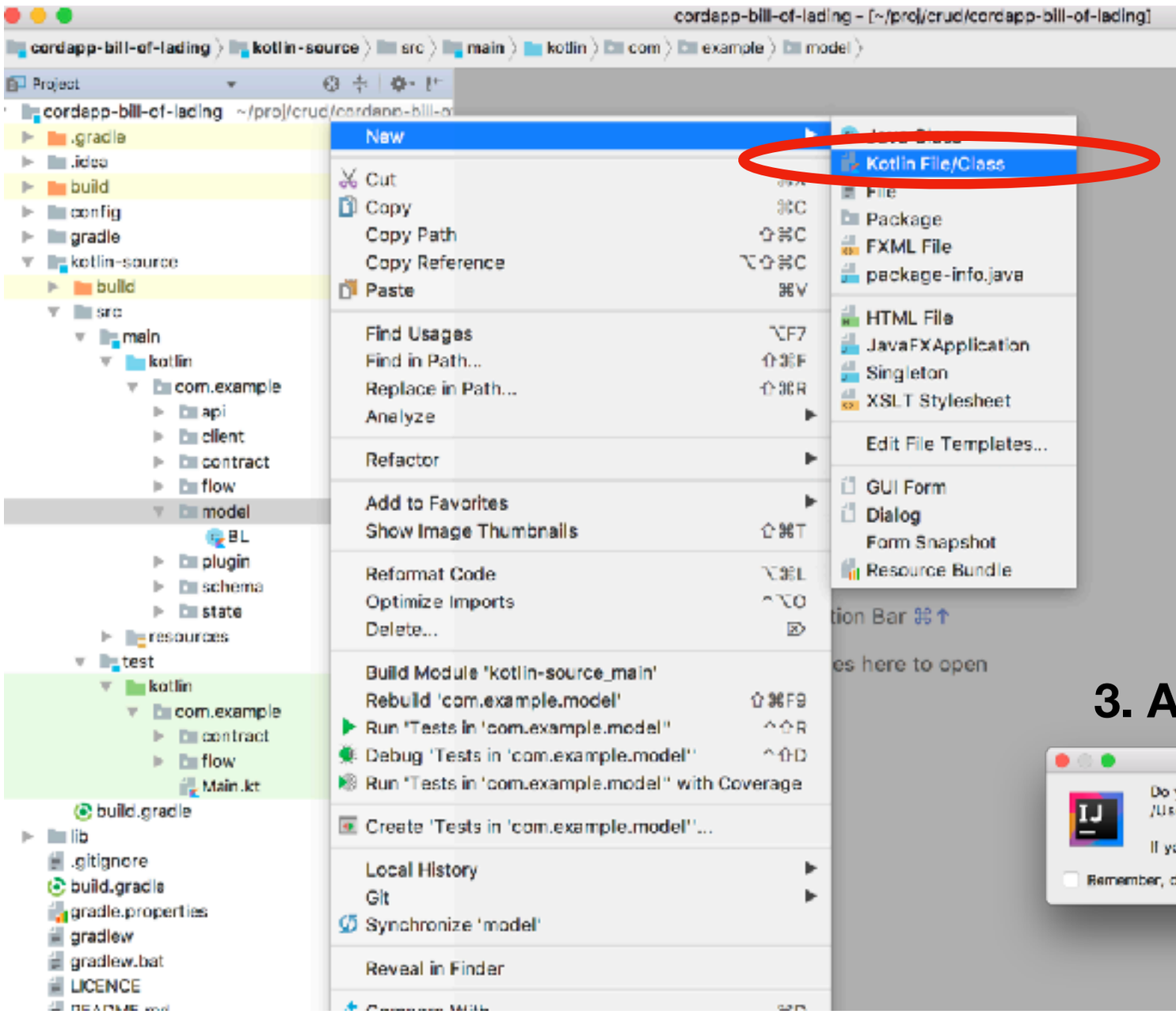
- Model
  Data structure for participants. REST API requires these fields for JSON data submission.

- Schema
  Column name definition for H2 database.

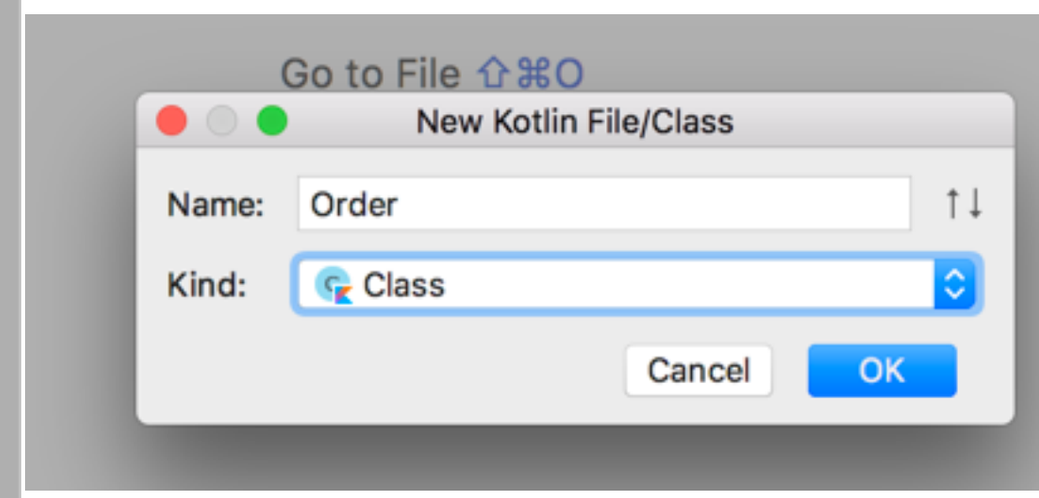
- State
  Data class for state, invoked the Model and Schema classes.

Create Order class in model

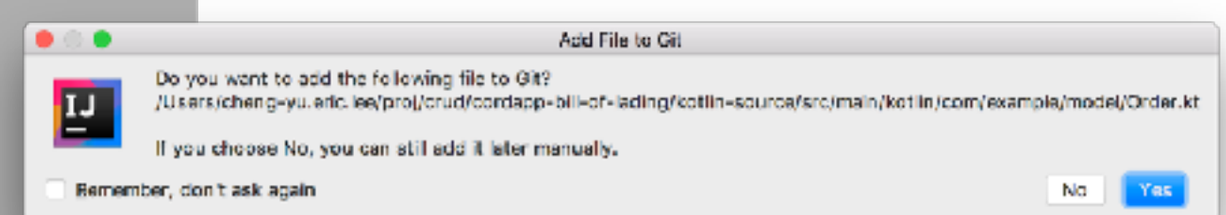
1. Right click on kotlin-source/src/main/kotlin/com.example/model ,
New > Kotlin Files/Class



2. Name is Order, Kind is Class.



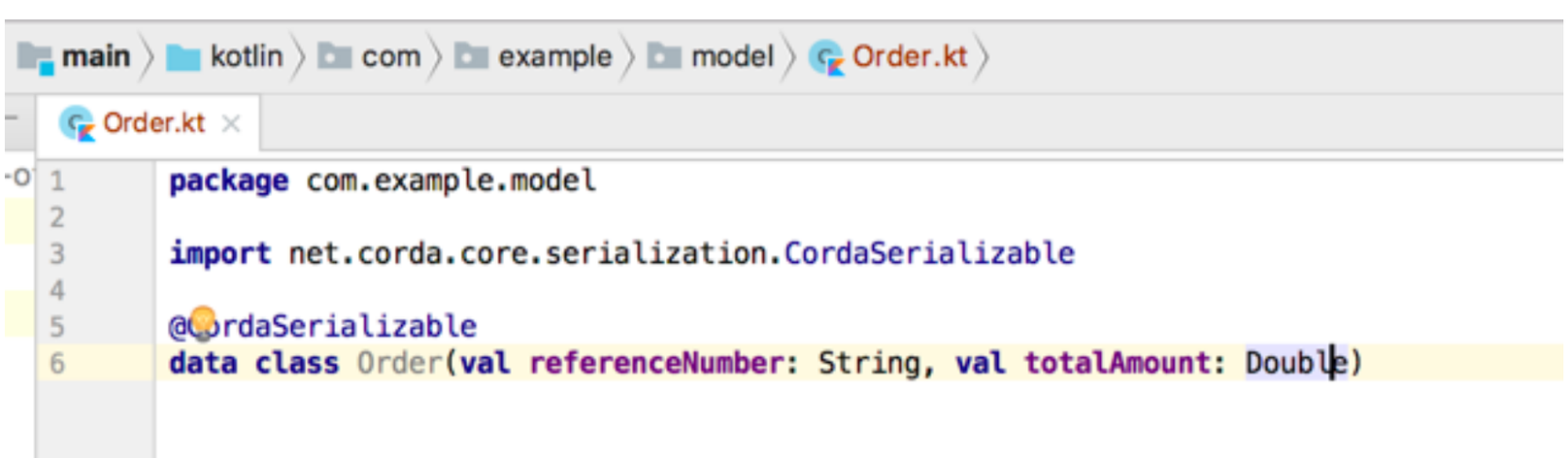
3. Add this new file(class) to git or not?



Edit the Code for Order.kt

```
package com.example.model

import net.corda.core.serialization.CordaSerializable

@CordaSerializable
data class Order(val referenceNumber: String, val totalAmount: Double)
```

Field name is referenceNumber, data type is string. **Field name is totalAmount, data type is double.**



Create the Code for Schema

- Create OrderSchema.kt under kotlin-source/src/main/kotlin/com.example/schema

```kotlin
package com.example.schema

import net.corda.core.schemas.MappedSchema
import net.corda.core.schemas.PersistentState
import java.time.Instant
import java.util.*
import javax.persistence.Column
import javax.persistence.Entity
import javax.persistence.Table

object OrderSchema

object OrderSchemaV1 : MappedSchema(
        schemaFamily = OrderSchema.javaClass,
        version = 1,
        mappedTypes = listOf(PersistentBL::class.java)) {
    @Entity
    @Table(name = "order_states")
    class PersistentBL(
            @Column(name = "seller_name")
            var sellerName: String,

            @Column(name = "buyer_name")
            var buyerName: String,

            @Column(name = "amount")
            var totalAmount: Double,

            @Column(name = "reference_no")
            var referenceNumber: String,

            @Column(name = "date")
            var date: Instant

    ) : PersistentState()
}
```

Table name and column name for H2 database.
We create a “order_states” table with five columns: seller_name, buyer_name, amount, reference_no and date.

Create the Code for State

- Create OrderState class under kotlin-source/src/main/kotlin/com.example/state

```kotlin
package com.example.state

import com.example.contract.OrderContract
import com.example.model.Order
import com.example.schema.OrderSchemaV1
import net.corda.core.contracts.ContractState
import net.corda.core.contracts.LinearState
import net.corda.core.contracts.UniqueIdentifier
import net.corda.core.identity.AbstractParty
import net.corda.core.identity.Party
import net.corda.core.schemas.MappedSchema
import net.corda.core.schemas.PersistentState
import net.corda.core.schemas.QueryableState
import net.corda.core.crypto.keys
import java.security.PublicKey
import java.time.Instant
import java.util.*

data class OrderState(val order: Order,
                      val seller: Party,
                      val buyer: Party,
                      val date: Instant = Instant.now(),
                      override val linearId: UniqueIdentifier = UniqueIdentifier()
): LinearState, QueryableState {

    override val contract get() = OrderContract()

    /** The public keys of the involved parties. */
    override val participants: List<AbstractParty> get() = listOf(buyer)

    /** Tells the vault to track a state if we are one of the parties involved. */
    override fun isRelevant(ourKeys: Set<PublicKey>) = ourKeys.intersect(participants.flatMap { it.owningKey.keys }).isNotEmpty()

    fun withNewOwner(newOwner: Party) = copy(buyer = newOwner)

    override fun generateMappedObject(schema: MappedSchema): PersistentState {
        return when (schema) {
            is OrderSchemaV1 -> OrderSchemaV1.PersistentBL(
                    sellerName = this.seller.name.toString(),
                    buyerName = this.buyer.name.toString(),
                    referenceNumber = this.order.referenceNumber,
                    totalAmount = this.order.totalAmount,
                    date = Instant.now()
            )
            else -> throw IllegalArgumentException("Unrecognised schema $schema")
        }
    }

    override fun supportedSchemas(): Iterable<MappedSchema> = listOf(OrderSchemaV1)
}
```

State variables and its data type.

Mapping the database schema to state variables.

Got Errors?

```
package com.example.state

import com.example.contract.OrderContract
import com.example.model.Order
import com.example.schema.OrderSchemaV1
import net.corda.core.contracts.ContractState
import net.corda.core.contracts.LinearState
import net.corda.core.contracts.UniqueIdentifier
import net.corda.core.identity.AbstractParty
import net.corda.core.identity.Party
import net.corda.core.schemas.MappedSchema
import net.corda.core.schemas.PersistentState
import net.corda.core.schemas.QueryableState
import net.corda.core.crypto.keys
import java.security.PublicKey
import java.time.Instant
import java.util.*


data class OrderState(val order: Order,
                      val seller: Party,
                      val buyer: Party,
                      val date: Instant = Instant.now(),
                      override val linearId: UniqueIdentifier = UniqueIdentifier()
): LinearState, QueryableState {

    override val contract get() = OrderContract()

    /** The public keys of the involved parties. */
    override val participants: List<AbstractParty> get() = listOf(buyer)

    /** Tells the vault to track a state if we are one of the parties involved. */
    override fun isRelevant(ourKeys: Set<PublicKey>) : Boolean = ourKeys.intersect(participants.flatMap { it.owningKey.keys }).isNotEmpty()

    fun withNewOwner(newOwner: Party) : OrderState = copy(buyer = newOwner)

    override fun generateMappedObject(schema: MappedSchema): PersistentState {
        return when (schema) {
            is OrderSchemaV1 -> OrderSchemaV1.PersistentBL(
                    sellerName = this.seller.name.toString()
```

Missing class/function, will create a “Contract"

Create the Contract Code

- Create OrderContract class under kotlin-source/src/main/kotlin/com.example/contract

```
package com.example.contract

import com.example.state.OrderState
import net.corda.core.contracts.*
import net.corda.core.crypto.SecureHash
import net.corda.core.transactions.LedgerTransaction

open class OrderContract : Contract {

    override fun verify(tx: LedgerTransaction) {
        val command = tx.commands.requireSingleCommand<Commands>()
        when (command.value) {
            is Commands.Issue -> {
                // Issuance verification logic.
                requireThat {
                    // Generic constraints around the order transaction.
                    "No inputs should be consumed when issuing an order." using (tx.inputs.isEmpty())
                    "Only one output state should be created." using (tx.outputs.size == 1)
                    val out = tx.outputsOfType<OrderState>().single()
                    "The seller and the buyer cannot be the same entity." using (out.seller != out.buyer)
                    "The amount cannot be 0." using (out.order.totalAmount != 0.0)
                    "The seller and the buyer must be signers." using (command.signers.containsAll(listOf(out.seller.owningKey, out.buyer.owningKey)))

                }
            }
            is Commands.Move -> {
                // Transfer verification logic.
                requireThat {
                    "Only one input should be consumed when move an order." using (tx.inputs.size == 1)
                    "Only one output state should be created." using (tx.outputs.size == 1)
                    val input = tx.inputsOfType<OrderState>().single()
                    val out = tx.outputsOfType<OrderState>().single()
                    "Buyer must be changed when move an order." using(input.buyer != out.buyer)
                }
            }
        }

    }

    /**
     * This contract implements commands: Issue, Move.
     */
    interface Commands : CommandData {
        class Issue : TypeOnlyCommandData(), Commands
        class Move : TypeOnlyCommandData(), Commands
    }

    /** This is a reference to the underlying legal contract template and associated parameters. */
    override val legalContractReference: SecureHash = SecureHash.sha256("contract template and params")
}
```

The rules for issue operation, using programming logic to represent the legal terms

The contract supports issue and transfer (move) operations.

For the legal contract reference, can put the file name, URL or etc.

Contract Details Example

```
// Issuance verification logic.
requireThat {
    // Generic constraints around the order transaction.
    "No inputs should be consumed when issuing an order." using (tx.inputs.isEmpty())
    "Only one output state should be created." using (tx.outputs.size == 1)
    val out = tx.outputsOfType<OrderState>().single()
    "The seller and the buyer cannot be the same entity." using (out.seller != out.buyer)
    "The amount cannot be 0." using (out.order.totalAmount != 0.0)
    "The seller and the buyer must be signers." using
(command.signers.containsAll(listOf(out.seller.owningKey, out.buyer.owningKey)))

}
```

- If no transaction input, cannot create the contract.
- After the contract execution, only one output.
- Cannot sell the goods/service to yourself.
- Transaction amount must greater than zero.
- Valid contract requires the signature from Seller and buyer.

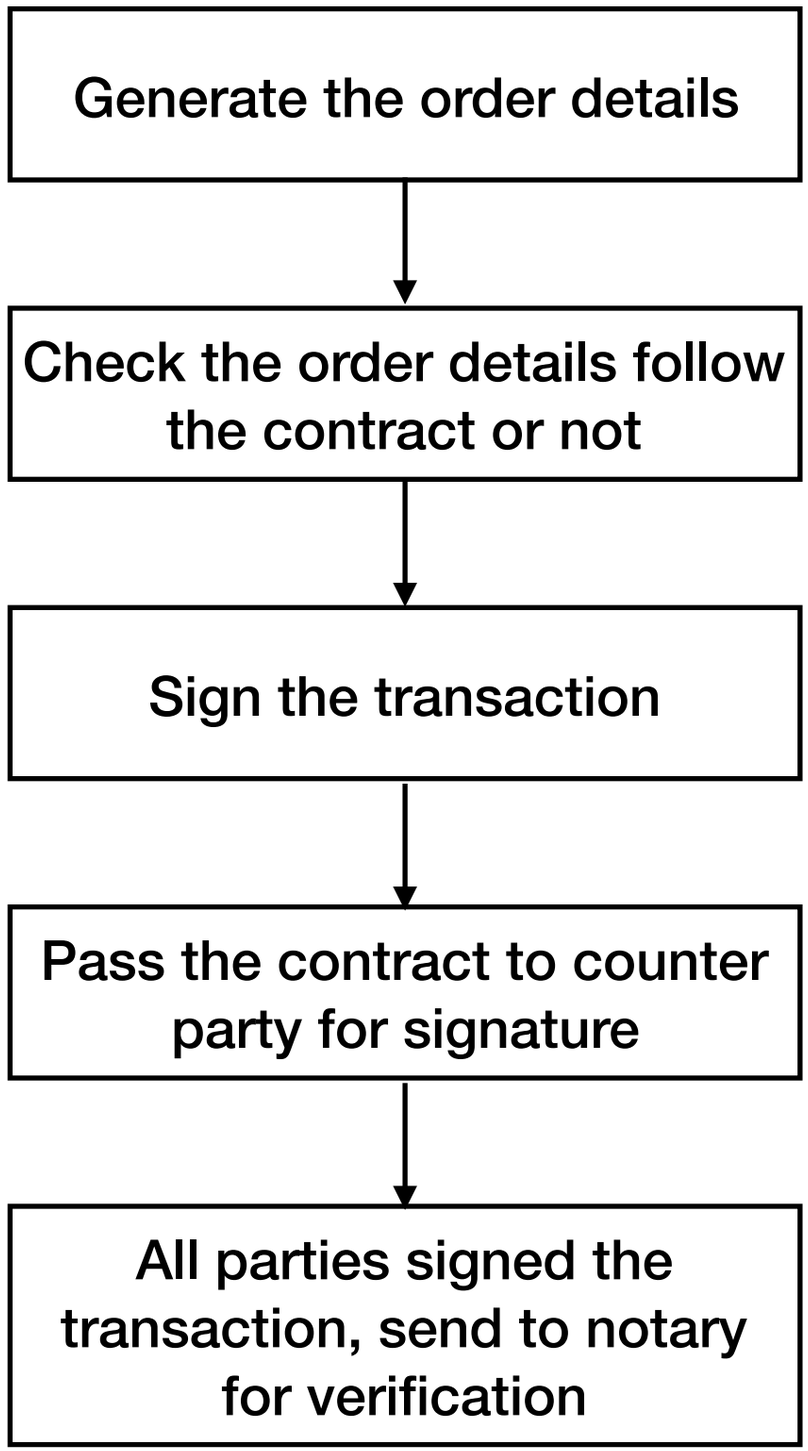
Error Disappear

Order.kt × OrderScheme.kt × OrderState.kt × OrderContract.kt ×

```kotlin
package com.example.state

import com.example.contract.OrderContract
import com.example.model.Order
import com.example.schema.OrderSchemaV1
import net.corda.core.contracts.ContractState
import net.corda.core.contracts.LinearState
import net.corda.core.contracts.UniqueIdentifier
import net.corda.core.identity.AbstractParty
import net.corda.core.identity.Party
import net.corda.core.schemas.MappedSchema
import net.corda.core.schemas.PersistentState
import net.corda.core.schemas.QueryableState
import net.corda.core.crypto.keys
import java.security.PublicKey
import java.time.Instant
import java.util.*

data class OrderState(val order: Order,
                      val seller: Party,
                      val buyer: Party,
                      val date: Instant = Instant.now(),
                      override val linearId: UniqueIdentifier = UniqueIdentifier()
): LinearState, QueryableState {

    override val contract : OrderContract  get() = OrderContract()

    /** The public keys of the involved parties. */
    override val participants: List<AbstractParty> get() = listOf(buyer)

    /** Tells the vault to track a state if we are one of the parties involved. */
    override fun isRelevant(ourKeys: Set<PublicKey>) : Boolean  = ourKeys.intersect(participants.flatMap { it.owningKey.keys }).isNotEmpty()

    fun withNewOwner(newOwner: Party) : OrderState  = copy(buyer = newOwner)

    override fun generateMappedObject(schema: MappedSchema): PersistentState {
        return when (schema) {
            is OrderSchemaV1 -> OrderSchemaV1.PersistentBL(
                    sellerName = this.seller.name.toString(),
                    buyerName = this.buyer.name.toString(),
                    referenceNumber = this.order.referenceNumber,
                    totalAmount = this.order.totalAmount,
                    date = Instant.now()
            )
            else -> throw IllegalArgumentException("Unrecognised schema $schema")
        }
    }

    override fun supportedSchemas(): Iterable<MappedSchema> = listOf(OrderSchemaV1)
}
```

Contract related errors fixed. (From red to black)
The contract controls the data commitment.

Create the Order Flow



Create the order details based on the required data defined by model and state.

Compare the order details and the contract, the value must match the contract rules.

If the order match the contract, sign it.

Counter party has to sign the transaction also.

All parties signed the transaction, the notary node verify it. If no error, commit to ledger.

Create the Order Flow Code

- Create CreateOrderFlow object under kotlin-source/src/main/kotlin/com.example/flow

```kotlin
package com.example.flow


import co.paralleluniverse.fibers.Suspendable
import com.example.contract.OrderContract
import com.example.flow.CreateOrderFlow.Acceptor
import com.example.flow.CreateOrderFlow.Initiator
import com.example.state.OrderState
import net.corda.core.contracts.Command
import net.corda.core.contracts.TransactionType
import net.corda.core.contracts.requireThat
import net.corda.core.flows.*
import net.corda.core.identity.Party
import net.corda.core.transactions.SignedTransaction
import net.corda.core.transactions.TransactionBuilder
import net.corda.core.utilities.ProgressTracker

object CreateOrderFlow {
    @InitiatingFlow
    @StartableByRPC
    class Initiator(val orderState: OrderState,
                    val otherParty: Party): FlowLogic<SignedTransaction>() {
        /**
         * The progress tracker checkpoints each stage of the flow and outputs the specified messages when each
         * checkpoint is reached in the code. See the 'progressTracker.currentStep' expressions within the call() function.
         */
        companion object {
            object GENERATING_TRANSACTION : ProgressTracker.Step("Generating transaction based on new order.")
            object VERIFYING_TRANSACTION : ProgressTracker.Step("Verifying contract constraints.")
            object SIGNING_TRANSACTION : ProgressTracker.Step("Signing transaction with our private key.")
            object GATHERING_SIGS : ProgressTracker.Step("Gathering the counterparty's signature.") {
                override fun childProgressTracker() = CollectSignaturesFlow.tracker()
            }
            object FINALISING_TRANSACTION : ProgressTracker.Step("Obtaining notary signature and recording transaction.") {
                override fun childProgressTracker() = FinalityFlow.tracker()
            }

            fun tracker() = ProgressTracker(
                    GENERATING_TRANSACTION,
                    VERIFYING_TRANSACTION,
                    SIGNING_TRANSACTION,
                    GATHERING_SIGS,
                    FINALISING_TRANSACTION
            )
        }

        override val progressTracker = tracker()

        /**
         * The flow logic is encapsulated within the call() method.
         */
        @Suspendable
        override fun call(): SignedTransaction {
            // Obtain a reference to the notary we want to use.
            val notary = serviceHub.networkMapCache.notaryNodes.single().notaryIdentity

            // Stage 1.
            progressTracker.currentStep = GENERATING_TRANSACTION
            // Generate an unsigned transaction.
            val txCommand = Command(OrderContract.Commands.Issue(), listOf(orderState.seller.owningKey, orderState.buyer.owningKey))
            val txBuilder = TransactionBuilder(TransactionType.General, notary).withItems(orderState, txCommand)

            // Stage 2.
            progressTracker.currentStep = VERIFYING_TRANSACTION
            // Verify that the transaction is valid.
            txBuilder.toWireTransaction().toLedgerTransaction(serviceHub).verify()

            // Stage 3.
            progressTracker.currentStep = SIGNING_TRANSACTION
            val partSignedTx = serviceHub.signInitialTransaction(txBuilder)

            // Stage 4.
            progressTracker.currentStep = GATHERING_SIGS
            // Send the state to the counterparty, and receive it back with their signature.
            val fullySignedTx = subFlow(CollectSignaturesFlow(partSignedTx, GATHERING_SIGS.childProgressTracker()))

            // Stage 5.
            progressTracker.currentStep = FINALISING_TRANSACTION
            // Notarise and record the transaction in both parties' vaults.
            return subFlow(FinalityFlow(fullySignedTx, FINALISING_TRANSACTION.childProgressTracker())).single()
        }
    }

    @InitiatedBy(Initiator::class)
    class Acceptor(val otherParty: Party) : FlowLogic<SignedTransaction>() {
        @Suspendable
        override fun call(): SignedTransaction {
            val signTransactionFlow = object : SignTransactionFlow(otherParty) {
                override fun checkTransaction(stx: SignedTransaction) = requireThat {
                    val output = stx.tx.outputs.single().data
                    "This must be an order transaction." using (output is OrderState)
                }
            }

            return subFlow(signTransactionFlow)
        }
    }
}
```

Required steps of the flow.

Steps implementation.

Discuss the flow implementation later.

Required Steps

```
companion object {
    object GENERATING_TRANSACTION : ProgressTracker.Step("Generating transaction based on new order.")
    object VERIFYING_TRANSACTION : ProgressTracker.Step("Verifying contract constraints.")
    object SIGNING_TRANSACTION : ProgressTracker.Step("Signing transaction with our private key.")
    object GATHERING_SIGS : ProgressTracker.Step("Gathering the counterparty's signature.") {
        override fun childProgressTracker() = CollectSignaturesFlow.tracker()
    }
    object FINALISING_TRANSACTION : ProgressTracker.Step("Obtaining notary signature and recording transaction.") {
        override fun childProgressTracker() = FinalityFlow.tracker()
    }

    fun tracker() = ProgressTracker(
            GENERATING_TRANSACTION,
            VERIFYING_TRANSACTION,
            SIGNING_TRANSACTION,
            GATHERING_SIGS,
            FINALISING_TRANSACTION
    )
}
```

Progress description for ProgressTracker

Progress item

Steps Logic Implementation

```
@Suspendable
override fun call(): SignedTransaction {
    // Obtain a reference to the notary we want to use.
    val notary = serviceHub.networkMapCache.notaryNodes.single().notaryIdentity

    // Stage 1.
    progressTracker.currentStep = GENERATING_TRANSACTION
    // Generate an unsigned transaction.
    val txCommand = Command(OrderContract.Commands.Issue(), listOf(orderState.seller.owningKey, orderState.buyer.owningKey))
    val txBuilder = TransactionBuilder(TransactionType.General, notary).withItems(orderState, txCommand)

    // Stage 2.
    progressTracker.currentStep = VERIFYING_TRANSACTION
    // Verify that the transaction is valid.
    txBuilder.toWireTransaction().toLedgerTransaction(serviceHub).verify()

    // Stage 3.
    progressTracker.currentStep = SIGNING_TRANSACTION
    val partSignedTx = serviceHub.signInitialTransaction(txBuilder)

    // Stage 4.
    progressTracker.currentStep = GATHERING_SIGS
    // Send the state to the counterparty, and receive it back with their signature.
    val fullySignedTx = subFlow(CollectSignaturesFlow(partSignedTx, GATHERING_SIGS.childProgressTracker()))

    // Stage 5.
    progressTracker.currentStep = FINALISING_TRANSACTION
    // Notarise and record the transaction in both parties' vaults.
    return subFlow(FinalityFlow(fullySignedTx, FINALISING_TRANSACTION.childProgressTracker())).single()
}
```

To initialise the transaction, need to know which transaction command and have a transaction builder.

Build the transaction and service hub check it by contract rules.

The party who proposed the transaction sign first.

Other party sign the transaction

Got all signatures, “FinalityFlow" will commit the transaction to ledger.

Code for Unit Test

- Create an OrderContractTest class under kotlin-source\src\test\kotlin\com.example\contract and paste the following codes

```
package com.example.contract

import com.example.flow.CreateOrderFlow
import com.example.model.Order
import com.example.state.OrderState
import net.corda.core.getOrThrow
import net.corda.testing.node.MockNetwork
import org.junit.After
import org.junit.Before
import org.junit.Test
import java.time.Instant
import java.util.*
import kotlin.test.assertEquals
import kotlin.test.assertTrue
import kotlin.test.fail

class OrderContractTest {
    lateinit var net: MockNetwork
    lateinit var a: MockNetwork.MockNode
    lateinit var b: MockNetwork.MockNode

    @Before
    fun setup() {
        net = MockNetwork()
        val nodes = net.createSomeNodes(3)
        a = nodes.partyNodes[0]
        b = nodes.partyNodes[1]
        // For real nodes this happens automatically, but we have to manually register the flow for tests
        nodes.partyNodes.forEach { it.registerInitiatedFlow(CreateOrderFlow.Acceptor::class.java) }
        net.runNetwork()
    }

    @After
    fun tearDown() {
        net.stopNodes()
    }

    @Test
    fun `Order amount must greater than zero`(){
        val state = OrderState(Order("a01",0.0),
                a.info.legalIdentity,
                b.info.legalIdentity,
                Instant.now()
        )
        try {
            val flow = CreateOrderFlow.Initiator(state, b.info.legalIdentity)
            val future = a.services.startFlow(flow).resultFuture
            net.runNetwork()
            val signedTx = future.getOrThrow()
            fail("No exception thrown!!")
        } catch(e:Exception){
            assertTrue(e.message.toString().contains(Regex("Contract verification failed: Failed requirement")))
        }

    }


}
```

Template for Corda Unit Test Code

- Mock network creation

Create two nodes (a and b) in Corda mock network

```
lateinit var net: MockNetwork
lateinit var a: MockNetwork.MockNode
lateinit var b: MockNetwork.MockNode

@Before
fun setup() {
    net = MockNetwork()
    val nodes = net.createSomeNodes(2)
    a = nodes.partyNodes[0]
    b = nodes.partyNodes[1]
    // For real nodes this happens automatically, but we have to manually register the flow
for tests
    nodes.partyNodes.forEach
{ it.registerInitiatedFlow(CreateOrderFlow.Acceptor::class.java) }
    net.runNetwork()
}
```

Start the nodes before testing

```
@After
fun tearDown() {
    net.stopNodes()
}
```

After testing, destroy the mock nodes

Unit Test Example

Story: I want to test if the order amount is zero

```
@Test
fun `Order amount must greater than zero`(){
    val state = OrderState(Order("a01",0.0),
            a.info.legalIdentity,
            b.info.legalIdentity,
            Instant.now()
    )
    try {
        val flow = CreateOrderFlow.Initiator(state, b.info.legalIdentity)
        val future = a.services.startFlow(flow).resultFuture
        net.runNetwork()
        val signedTx = future.getOrThrow()
        fail("No exception thrown!!")
    } catch(e:Exception){
        assertTrue(e.message.toString().contains(Regex("Contract verification failed:
Failed requirement")))
    }
}
```

Propose an order, a have a deal with b, the amount is 0

Start the transaction

If the transaction done, judge the test is fail.

With an exception is positive, the error has this message string pattern.

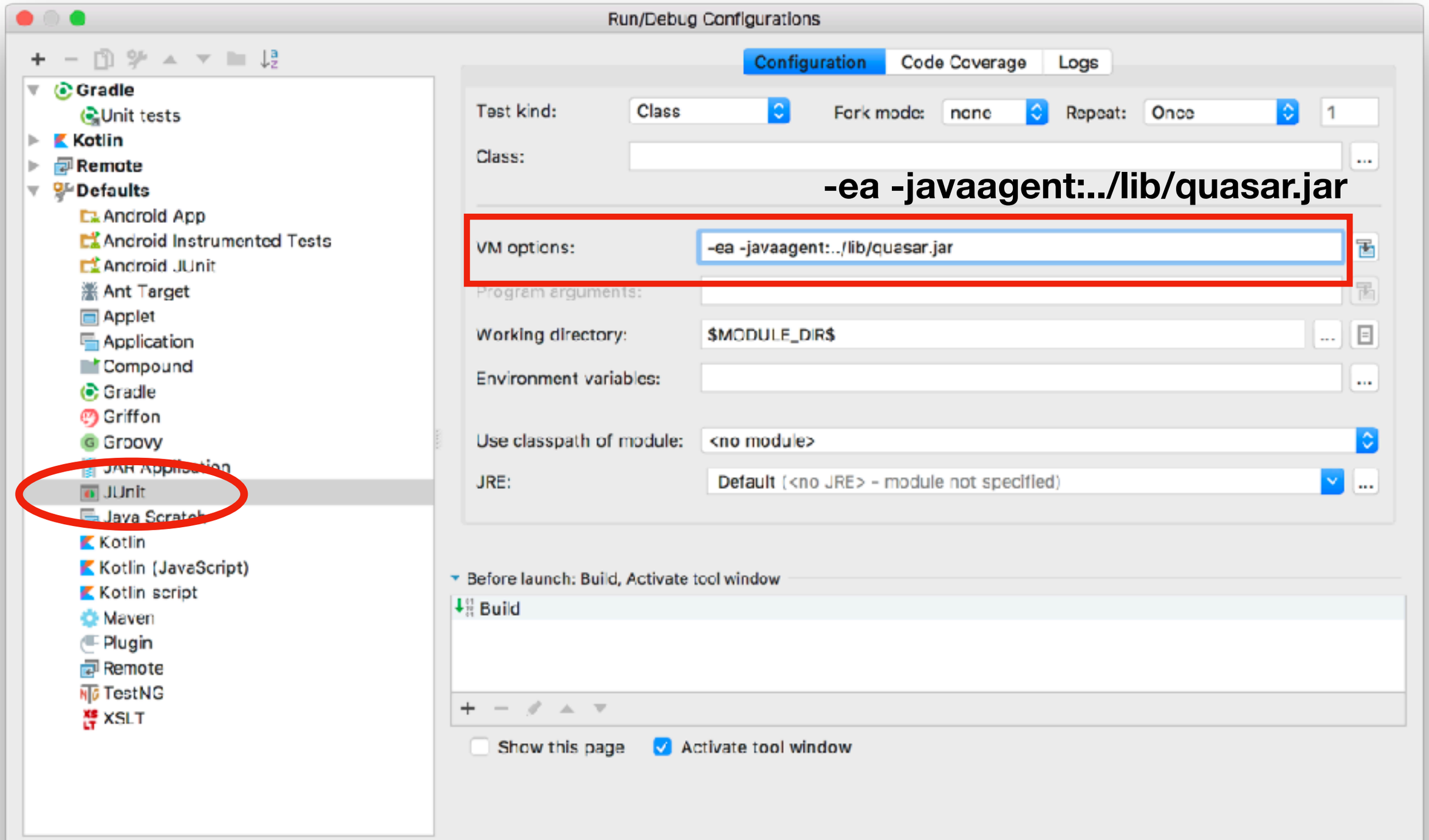
Run the test

- Right click on the OrderContractTest class, click “Run OrderContractTest”.
- The test should failed with the error message as the screenshot shown.

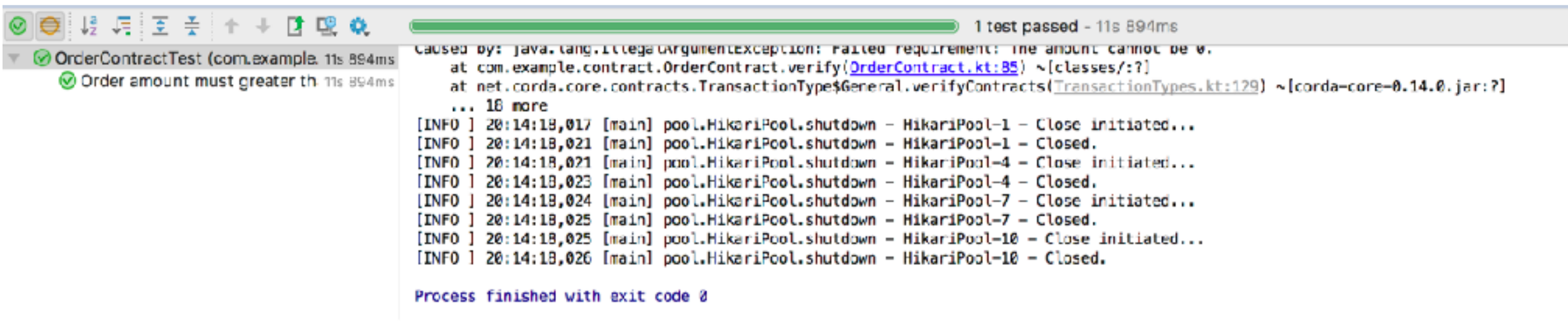
```
net.corda.node.internal.AbstractNode$ServiceHubInternalImpl.startFlow(AbstractNode.kt:812)
net.corda.node.services.api.ServiceHubInternal$DefaultImpls.startFlow(ServiceHubInternal.kt:134)
net.corda.node.internal.AbstractNode$ServiceHubInternalImpl.startFlow(AbstractNode.kt:772)
com.example.flow.OrderFlowTest.order flow records a transaction in the ledger of seller and buyer(OrderFlowTest.kt:67) <24 internal call
by: java.lang.IllegalArgumentException: Fiber class net.corda.node.services.statemachine.FlowStateMachineImpl has not been instrumented.
co.paralleluniverse.fibers.Fiber.<init>(Fiber.java:191)
co.paralleluniverse.fibers.Fiber.<init>(Fiber.java:436)
net.corda.node.services.statemachine.FlowStateMachineImpl.<init>(FlowStateMachineImpl.kt:36)
net.corda.node.services.statemachine.StateMachineManager.createFiber(StateMachineManager.kt:392)
net.corda.node.services.statemachine.StateMachineManager.access$createFiber(StateMachineManager.kt:74)
net.corda.node.services.statemachine.StateMachineManager$add$fiber$1.invoke(StateMachineManager.kt:471)
net.corda.node.services.statemachine.StateMachineManager$add$fiber$1.invoke(StateMachineManager.kt:74)
net.corda.node.utilities.CordaPersistence.inTopLevelTransaction(CordaPersistence.kt:67)
net.corda.node.utilities.CordaPersistence.transaction(CordaPersistence.kt:58)
```

Add VM Option

Run > Edit Configuration



Test Pass



You can create unit test cases, the code structure/style similar to jUnit.

Create REST API

- Add the following codes to kotlin-source/src/main/kotlin/com.example/api/ExampleApi.kt
  - Retrieve all orders (GET method)

```
@GET
@Path("orders")
@Produces(MediaType.APPLICATION_JSON)
fun getOrders(): List<StateAndRef<OrderState>>{
    val vaultStates = services.vaultQueryBy<OrderState>()
    return vaultStates.states
}
```

  - Create Order (PUT method)

```
@PUT
@Path("{seller}/{buyer}/create-order")
fun createOrder(order: Order,
                @PathParam("seller") seller: X500Name,
                @PathParam("buyer") buyer: X500Name) : Response{
    val seller =services.partyFromX500Name(seller)
    if (seller == null){
        return Response.status(Response.Status.BAD_REQUEST).build()
    }

    val buyer = services.partyFromX500Name(buyer)
    if (buyer == null){
        return Response.status(Response.Status.BAD_REQUEST).build()
    }

    val state = OrderState(
            order,
            seller,
            buyer,
            date = Instant.now()
    )

    val (status, msg) = try {
        val flowHandle = services.
                startTrackedFlowDynamic(CreateOrderFlow.Initiator::class.java, state, seller)
        flowHandle.progress.subscribe { println(">> $it") }

        val result = flowHandle.returnValue.getOrThrow()

        Response.Status.CREATED to "Transaction id ${result.id} committed to ledger."

    } catch (ex: Throwable){
        logger.error(ex.message, ex)
        Response.Status.BAD_REQUEST to "Transaction failed."
    }

    return Response.status(status).entity(msg).build()

}
```

Build Project

- Open the terminal for project in IntelliJ, execute the three commands
  - `./gradlew clean`
  - `./gradlew build`
  - `./gradlew deployNodes`

```
Terminal
ANAC02TP3F4HTD8:cordapp-bill-of-lading cheng-yu.eric.lee$ pwd
/Users/cheng-yu.eric.lee/proj/crud/cordapp-bill-of-lading
ANAC02TP3F4HTD8:cordapp-bill-of-lading cheng-yu.eric.lee$ ls
LICENCE          TRADEMARK        build.gradle     gradle           gradlew          kotlin-source    settings.gradle
README.md        build            config           gradle.properties gradlew.bat     lib
ANAC02TP3F4HTD8:cordapp-bill-of-lading cheng-yu.eric.lee$ ./gradlew clean
Starting a Gradle Daemon (subsequent builds will be faster)
:kotlin-source:clean

BUILD SUCCESSFUL

Total time: 6.107 secs
ANAC02TP3F4HTD8:cordapp-bill-of-lading cheng-yu.eric.lee$ ./gradlew build
33:6  LF:  UTF-8:  Git: master:
```

Deployment in Dev Mode

- Change working directory to {your project folder}/kotlin-source/build/nodes

- runnodes (Windows: runnodes.bat Others: runnodes)

1. Run the runnodes script

```
Terminal
AMAC02TP3F4HTD8:nodes cheng-yu.eric.lee$ pwd
/Users/cheng-yu.eric.lee/proj/crud/cordapp-bill-of-lading/kotlin-source/build/nodes
AMAC02TP3F4HTD8:nodes cheng-yu.eric.lee$ ls
Controller      NodeA           NodeB           NodeC           runnodes        runnodes.bat    runnodes.jar
AMAC02TP3F4HTD8:nodes cheng-yu.eric.lee$ sh runnodes
```

2. Starting the 4 nodes in terminal or CMD windows

```
cheng-yu.eric.lee — java ◂ bash -c cd /Users/cheng-yu.eric.lee/proj/crud/cordap...
-bash   -bash ...   java...& exit ...   java...& exit ...   jav...exit ... >>
Last login: Wed Aug 30 15:51:25 on ttys010
AMAC02TP3F4HTD8:~ cheng-yu.eric.lee$ bash -c 'cd /Users/cheng-yu.eric.lee/proj/c
rud/cordapp-bill-of-lading/kotlin-source/build/nodes/NodeC; /Library/Java/JavaVi
rtualMachines/jdk1.8.0_131.jdk/Contents/Home/jre/bin/java -Dname=NodeC-corda-web
server.jar -Dcapsule.jvm.args=-agentlib:jdwp=transport=dt_socket,server=y,suspen
d=n,address=5012 -jar corda-webserver.jar && exit'
objc[33887]: Class JavaLaunchHelper is implemented in both /Library/Java/JavaVir
tualMachines/jdk1.8.0_131.jdk/Contents/Home/jre/bin/java (0x1033894c0) and /Libr
ary/Java/JavaVirtualMachines/jdk1.8.0_131.jdk/Contents/Home/jre/lib/libinstrumen
t.dylib (0x10544a4e0). One of the two will be used. Which one is undefined.
Listening for transport dt_socket at address: 5012
Logs can be found in /Users/cheng-yu.eric.lee/proj/crud/cordapp-bill-of-lading/k
otlin-source/build/nodes/NodeC/logs/web
Starting as webserver: localhost:10013
```

depolyNodes Configuration File

- kotlin-source/build.gradle

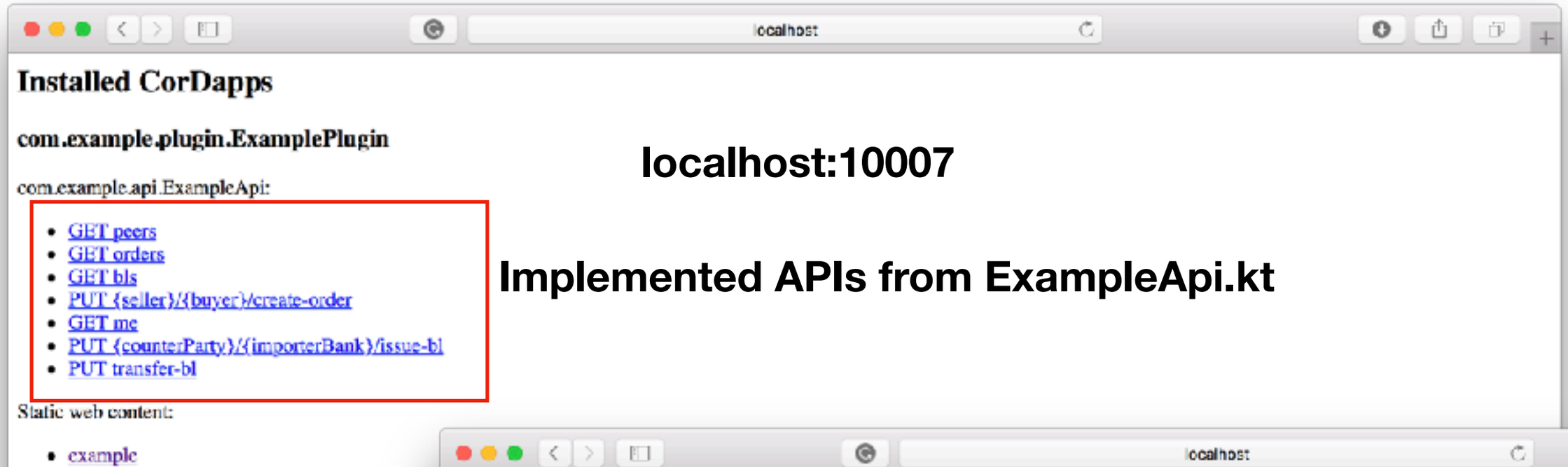
```
task deployNodes(type: net.corda.plugins.Cordform, dependsOn: ['jar']) {
    directory "./build/nodes"
    networkMap "CN=Controller,O=R3,OU=corda,L=London,C=UK"
    node {
        name "CN=Controller,O=R3,OU=corda,L=London,C=UK"
        advertisedServices = ["corda.notary.validating"]
        p2pPort 10002
        rpcPort 10003
        webPort 10004
        cordapps = []
    }
    node {
        name "CN=NodeA,O=NodeA,L=London,C=UK"
        advertisedServices = []
        p2pPort 10005
        rpcPort 10006
        webPort 10007
        cordapps = []
        rpcUsers = [[ user: "user1", "password": "test", "permissions": []]]
    }
    node {
        name "CN=NodeB,O=NodeB,L=New York,C=US"
        advertisedServices = []
        p2pPort 10008
        rpcPort 10009
        webPort 10010
        cordapps = []
        rpcUsers = [[ user: "user1", "password": "test", "permissions": []]]
    }
    node {
        name "CN=NodeC,O=NodeC,L=Paris,C=FR"
        advertisedServices = []
        p2pPort 10011
        rpcPort 10012
        webPort 10013
```

Node name (X.509 format)

Port numbers for different purposes.

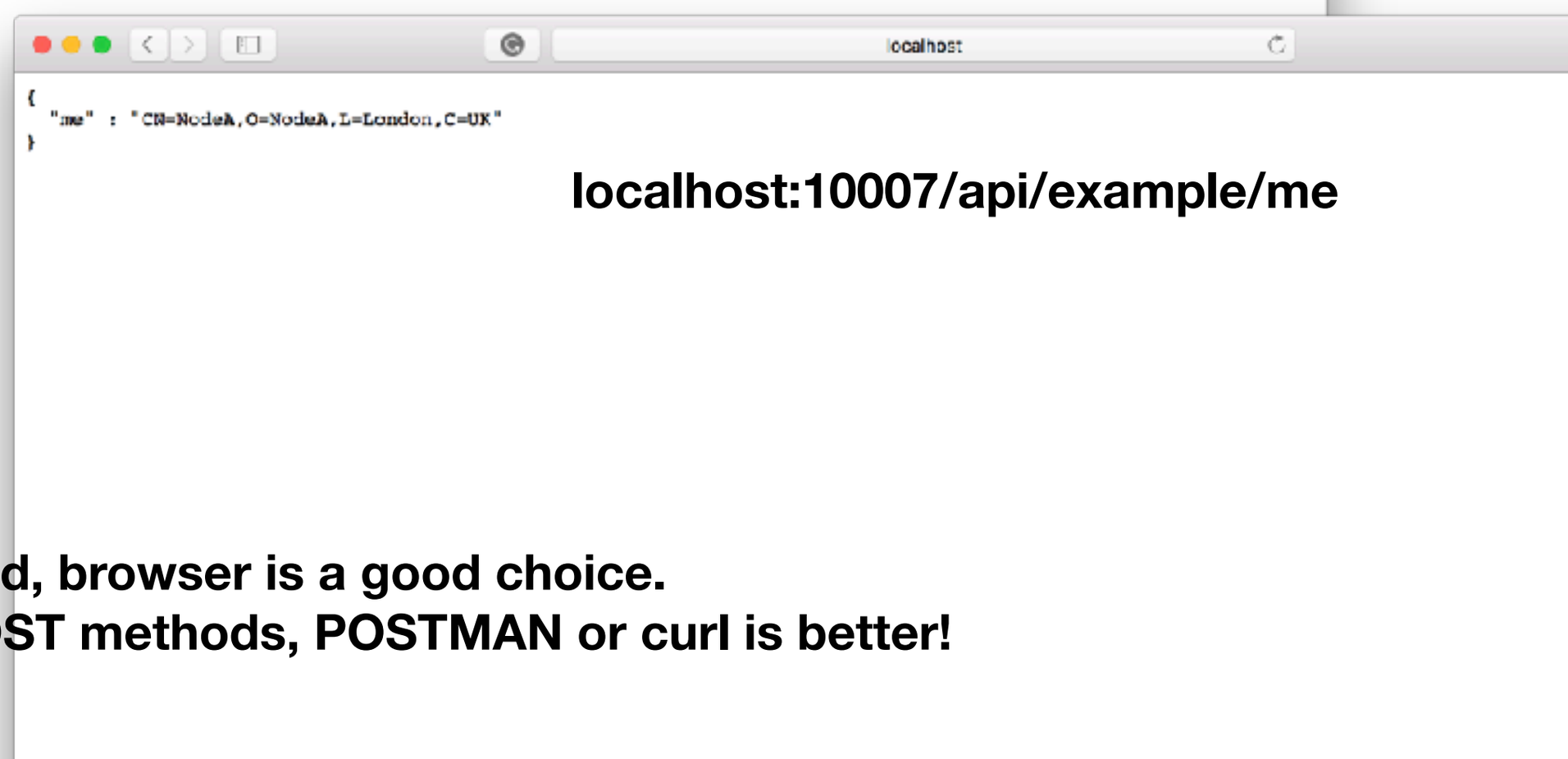
The configuration file defined the nodes' name and ports for communication.

Testing with Browser



localhost:10007

Implemented APIs from ExampleApi.kt

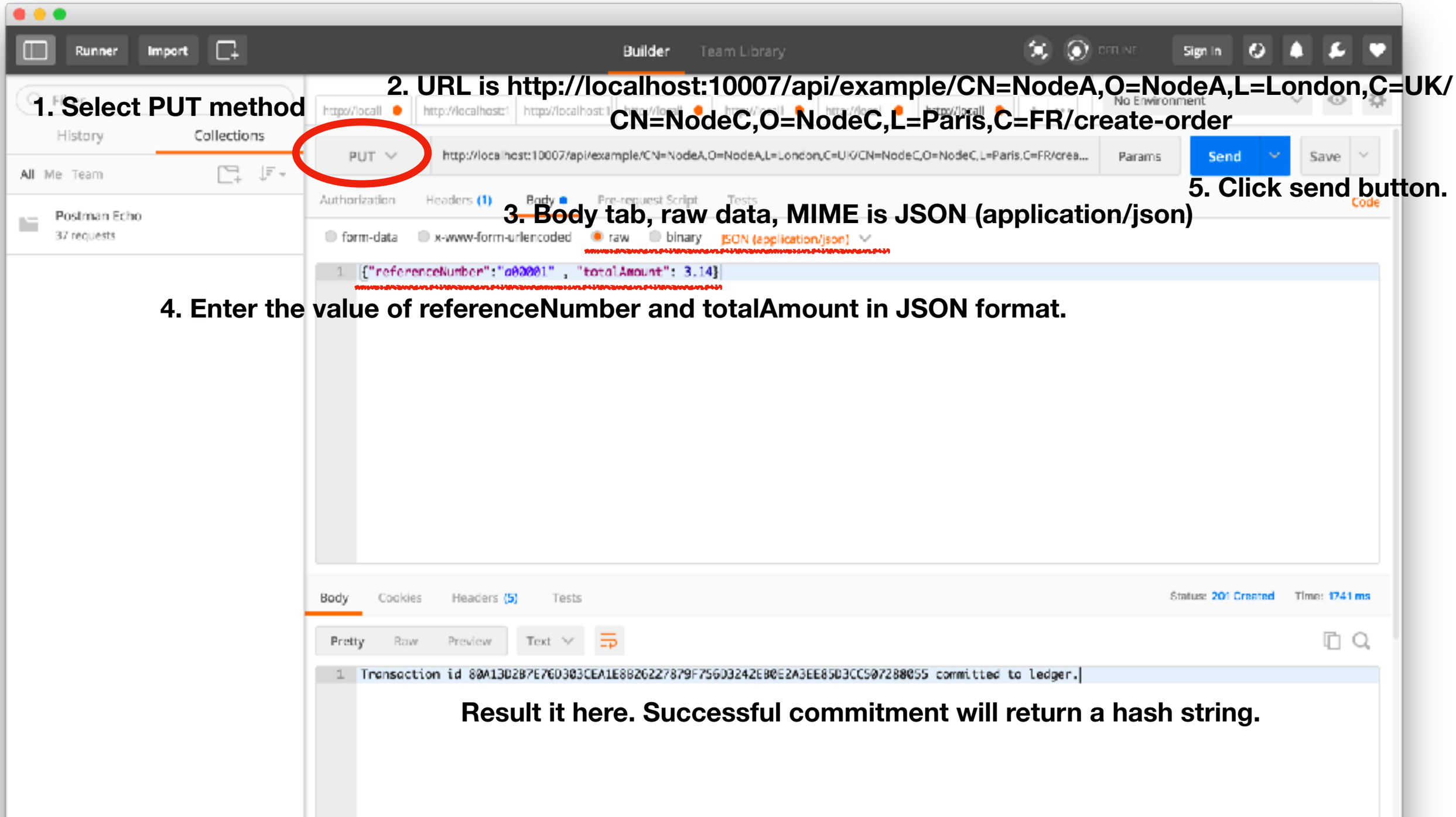


localhost:10007/api/example/me

Test “GET” method, browser is a good choice.
But for PUT or POST methods, POSTMAN or curl is better!

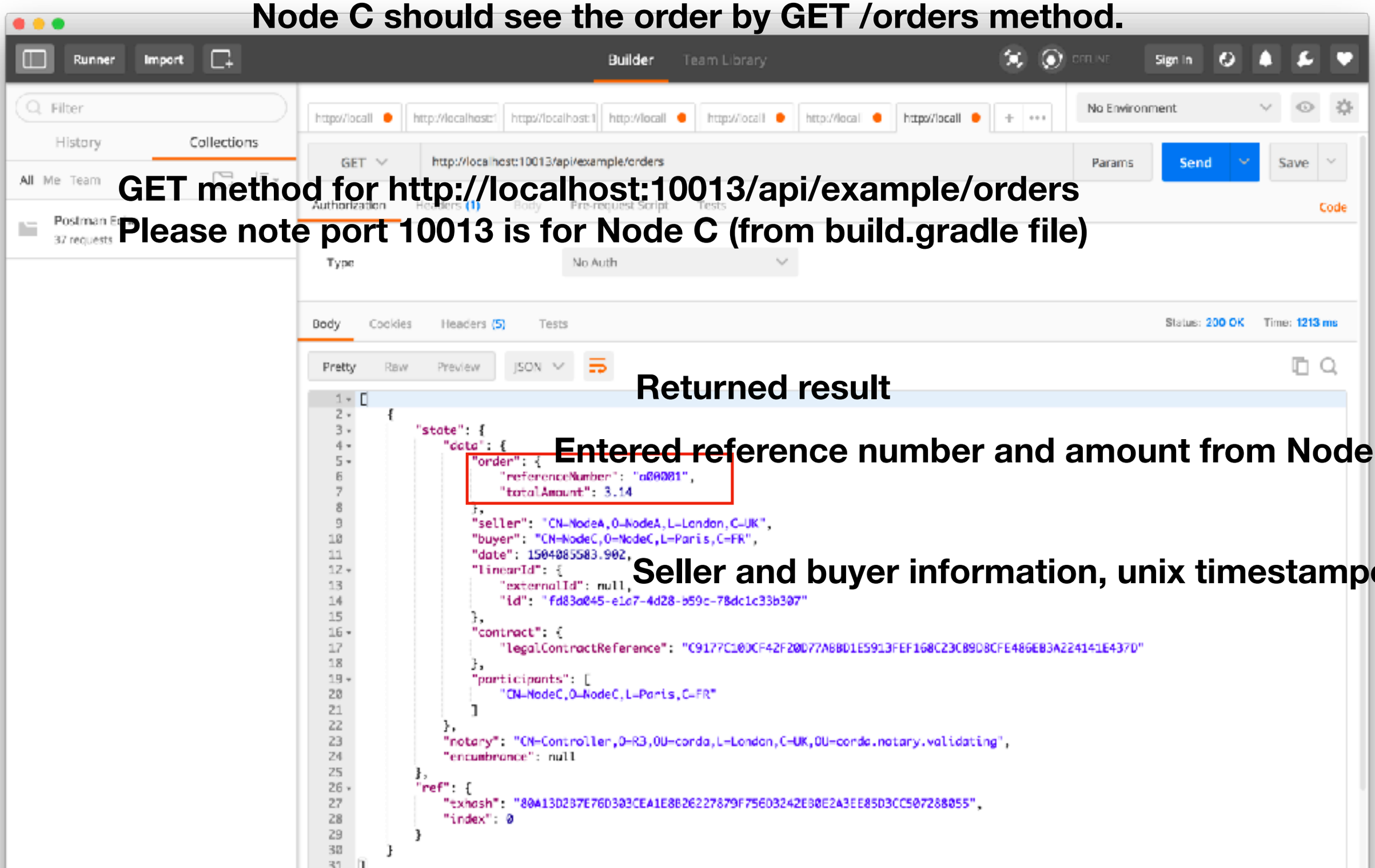
Testing with POSTMAN

Node A (seller) create a order with Node C (buyer), the order number is a00001, amount is 3.14



Validating the Result

Node C should see the order by GET /orders method.



GET method for http://localhost:10013/api/example/orders
Please note port 10013 is for Node C (from build.gradle file)

Returned result

Entered reference number and amount from Node A.

Seller and buyer information, unix timestamped.