Chapter 7: Seller Support and User Experience

Learning Bluemix & Blockchain

Bob Dill, IBM Distinguished Engineer, CTO Global Technical Sales
David Smits, Senior Certified Architect, IBM Blockchain

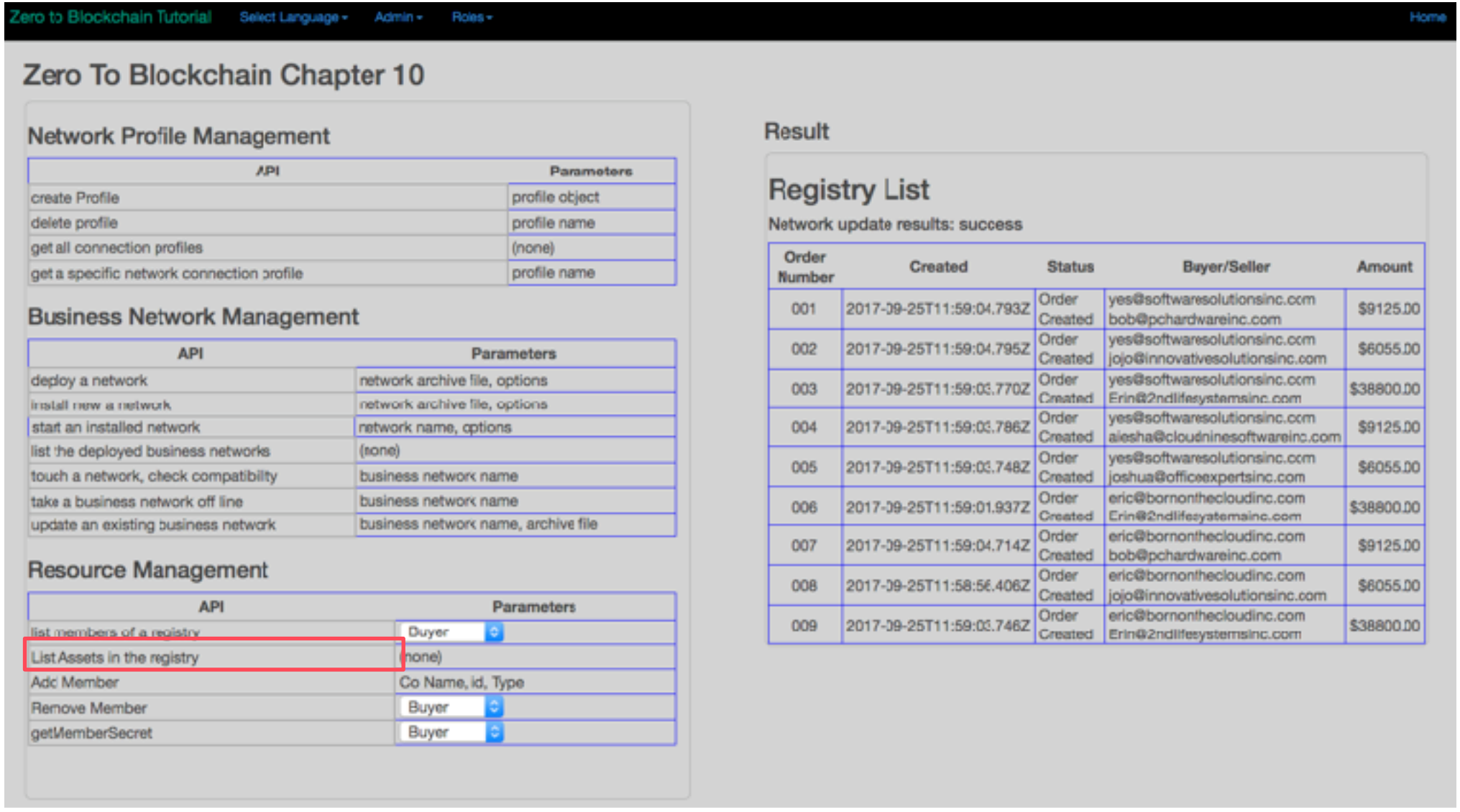
The Plan: 30 minute Chapters with an hour or two of practice

Chapter 1:	What is Blockchain? Concept and Architecture overview
Chapter 2:	What's the story we're going to build
Chapter 2.1:	Architecture for the Story
Chapter 3:	Set up local HyperLedger V1 development environment
Chapter 4:	Build and test the network
Chapter 5:	Administration User Experience
Chapter 6:	Buyer Support and User Experience
Chapter 7:	Seller Support and User Experience
Chapter 8:	Shipper Support and User Experience
Chapter 9:	Provider Support and User Experience
Chapter 10:	Finance Company Support and User Experience
Chapter 11:	Combining for Demonstration
Chapter 12:	Events and Automating for Demonstration

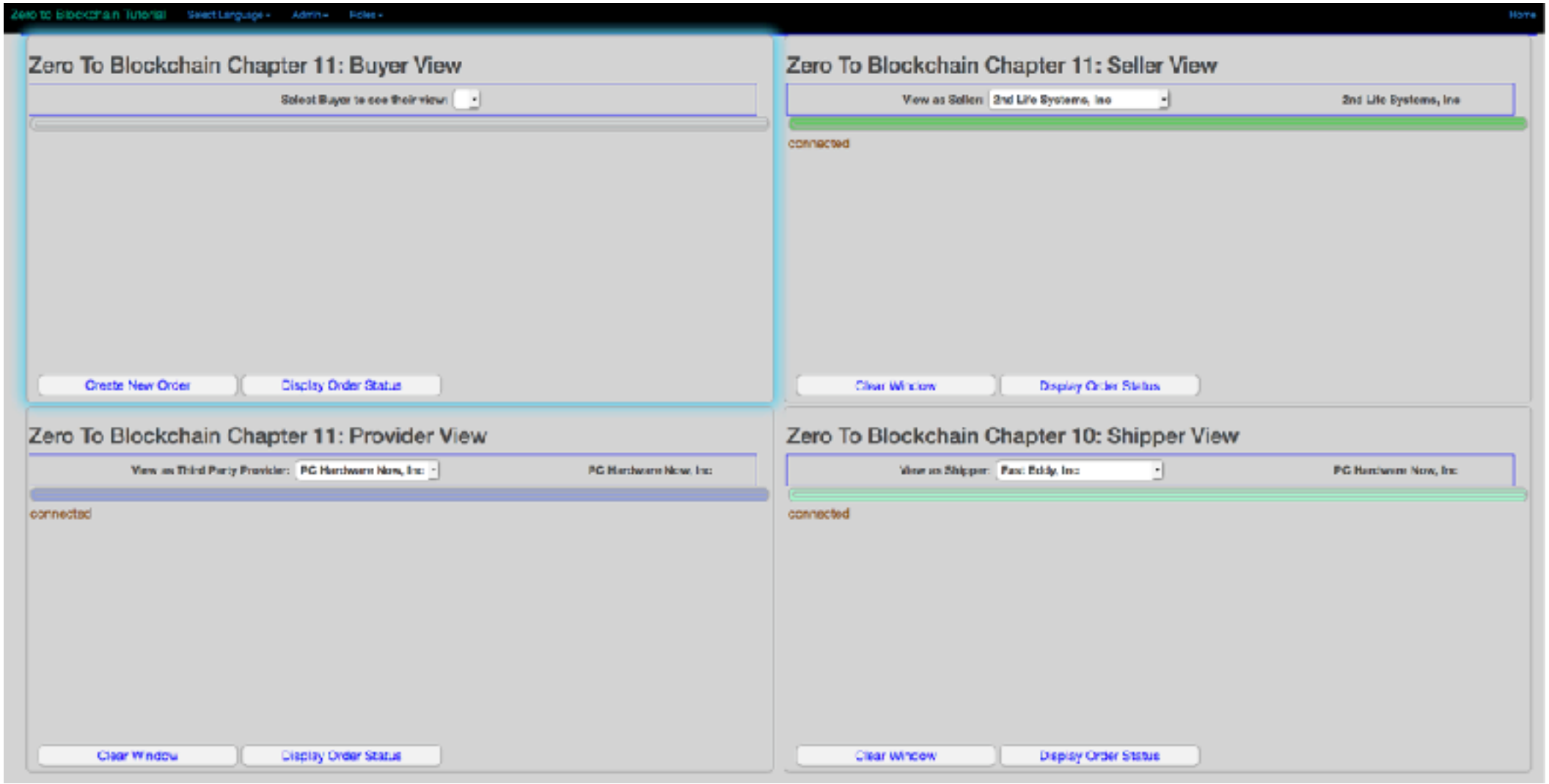
A couple of starting comments

- Each time you start, or restart docker or your docker containers, they start up in their original, unpopulated state. This means that they have no data already in the system. This happens when you run the ./startup.sh —or— buildAndDeploy scripts.
- The tutorial has an 'autoload' feature as part of the admin menu at the top of the screen. This will automatically create 20 members, 9 orders and the item file used during the 'create new order' process for the buyer.
- This command needs to be run if you want to prepopulate your systems with some data. We will look at adding transactions to this autoload process in the final chapter.
- Autoload will display its status, shown to the right, via a web socket channel.

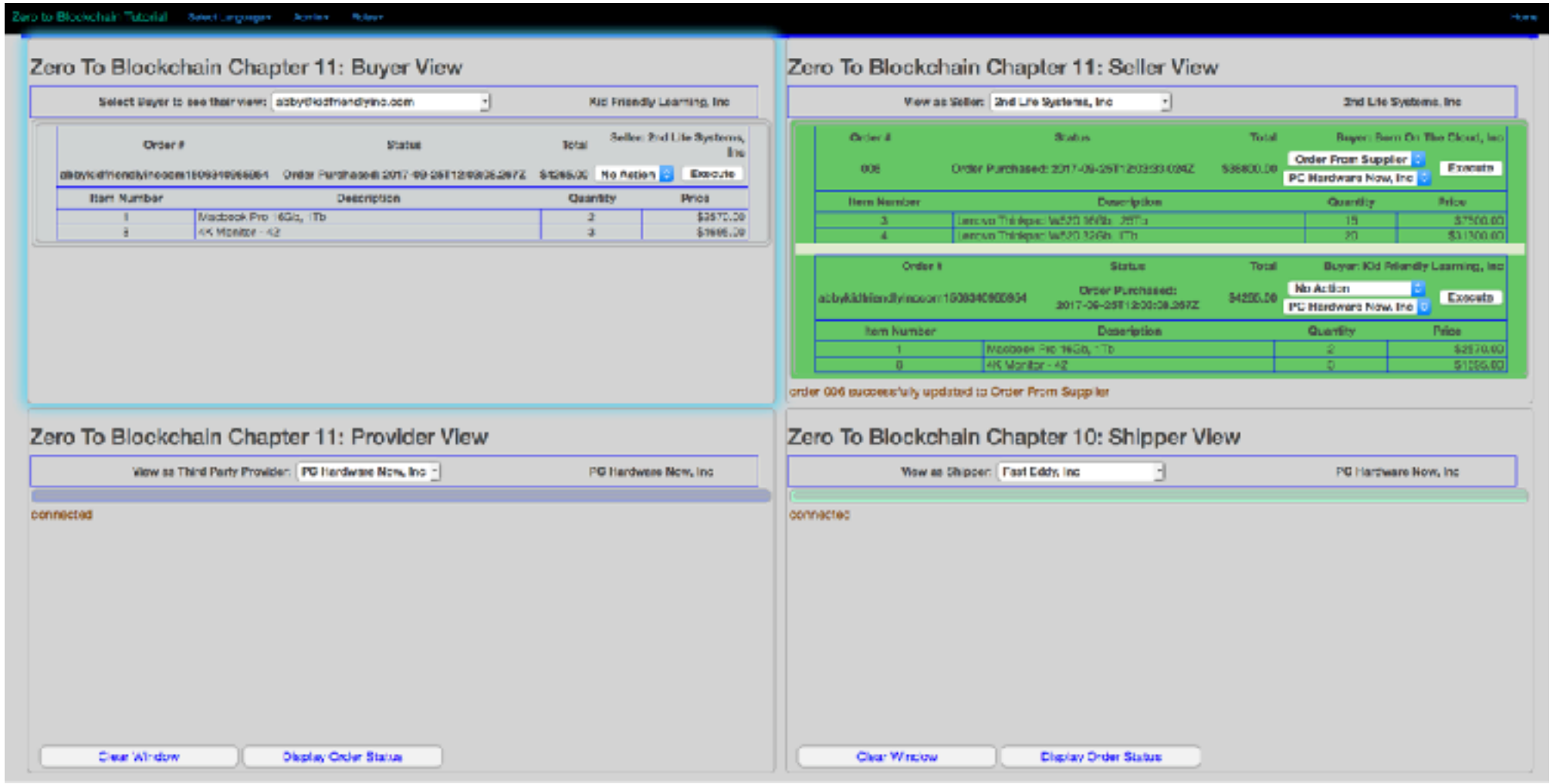
You can check the status of all orders in the system via the admin interface we built in Chapter 5



Over the next three lessons, we will build each of the panels in this picture. Today, we're focused on the top right: Seller



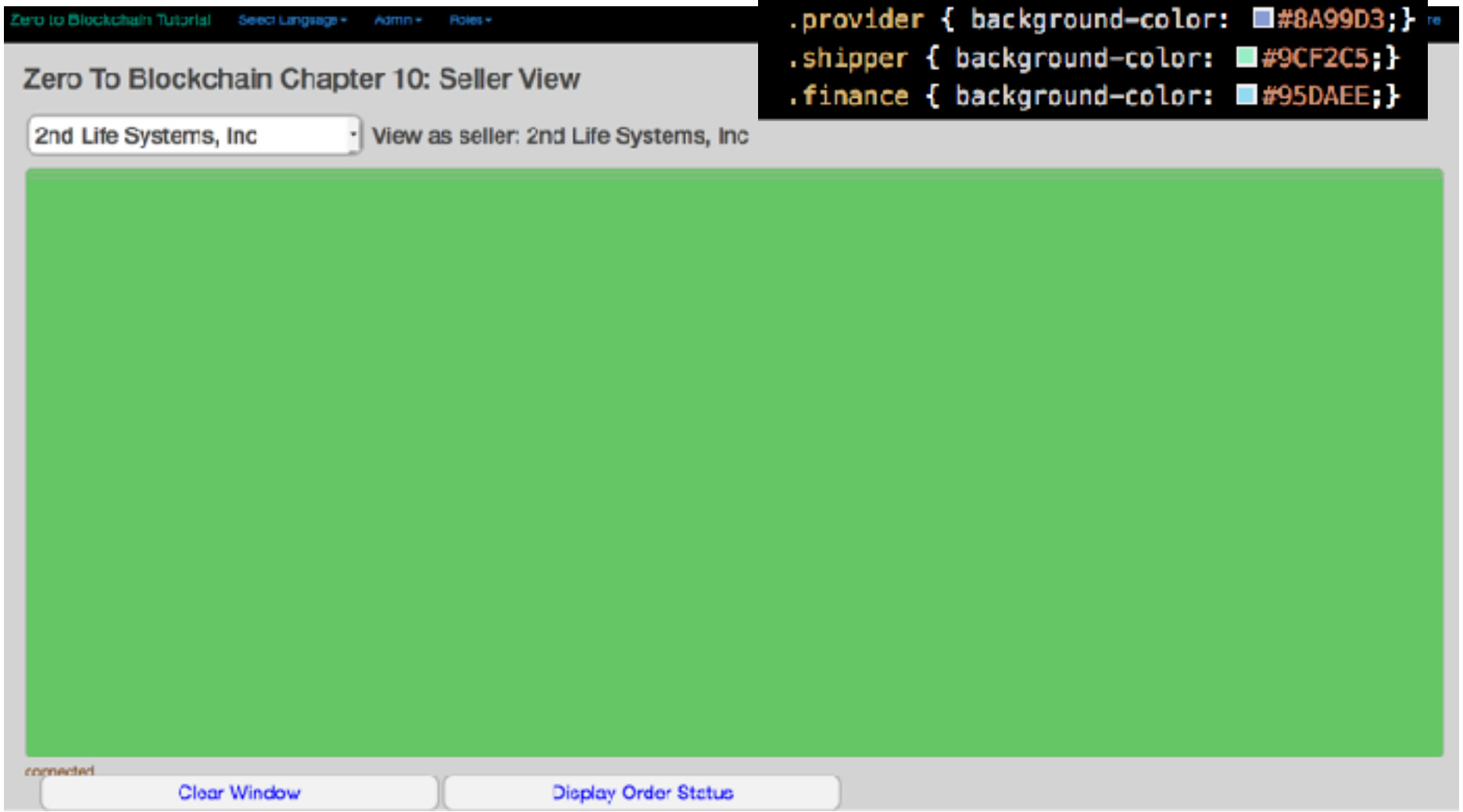
In Chapter 6, we created the buyer view, which allowed us to create orders, display orders, and selectively update the status of orders.
Today, we'll create the user experience for the Seller, to allow them to selectively update the status of an order.



You'll note that the Seller view has a green background. You can specify the color in the pageStyles.css file. This is done so that when all 4 roles are on the same browser page, it's easier to differentiate between them.

- We will reuse almost all of the code from the Buyer user experience to create the seller user experience.

```
.buyer { background-color: #CBCFD0; }
.seller { background-color: #5DC15D;}
.provider { background-color: #8A99D3;}
.shipper { background-color: #9CF2C5;}
.finance { background-color: #95DAEE;}
```



We built a query.qry file in the last session, which has the following query in it:

```
query selectOrders {
  description: "Select all Orders"
  statement:
      SELECT org.acme.Z2BTestNetwork.Order
}
```

- How does Hyperledger-Composer use the same query for different people and roles? Through the ACL definitions.

- We will do the same thing for the Seller, which will both enable access (if we do nothing, the Seller has absolutely no access) and also limit access to specific transactions.

```
rule BuyerACLCreate {
    description: "Enable Buyers to execute all actions on an O
    participant(m): "org.acme.Z2BTestNetwork.Buyer"
    operation: READ, CREATE, UPDATE
    resource(v): "org.acme.Z2BTestNetwork.**"
    transaction(tx): "org.acme.Z2BTestNetwork.CreateOrder"
    condition: (v.buyer.buyerID == m.getIdentifier())
    action: ALLOW
}
rule BuyerACLBuy {
    description: "Enable a Buyer to update an Order from Creat
    participant(m): "org.acme.Z2BTestNetwork.Buyer"
    operation: READ, CREATE, UPDATE
    resource(v): "org.acme.Z2BTestNetwork.**"
    transaction(tx): "org.acme.Z2BTestNetwork.Buy"
    condition: (v.buyer.buyerID == m.getIdentifier())
    action: ALLOW
}
rule BuyerACLCancel {
    description: "Enable a Buyer to CANCEL an Order"
    participant(m): "org.acme.Z2BTestNetwork.Buyer"
    operation: READ, CREATE, UPDATE, DELETE
    resource(v): "org.acme.Z2BTestNetwork.**"
    transaction(tx): "org.acme.Z2BTestNetwork.OrderCancel"
    condition: (v.buyer.buyerID == m.getIdentifier())
    action: ALLOW
}
rule BuyerACLDispute {
    description: "Enable a Buyer to raise a DISPUTE on an Orde
```

In Chapter 6, we created the OrderAction function, which allowed us to process order status changes for the Buyer. We will extend the set of options in that function to include the seller.

Buyer capability:

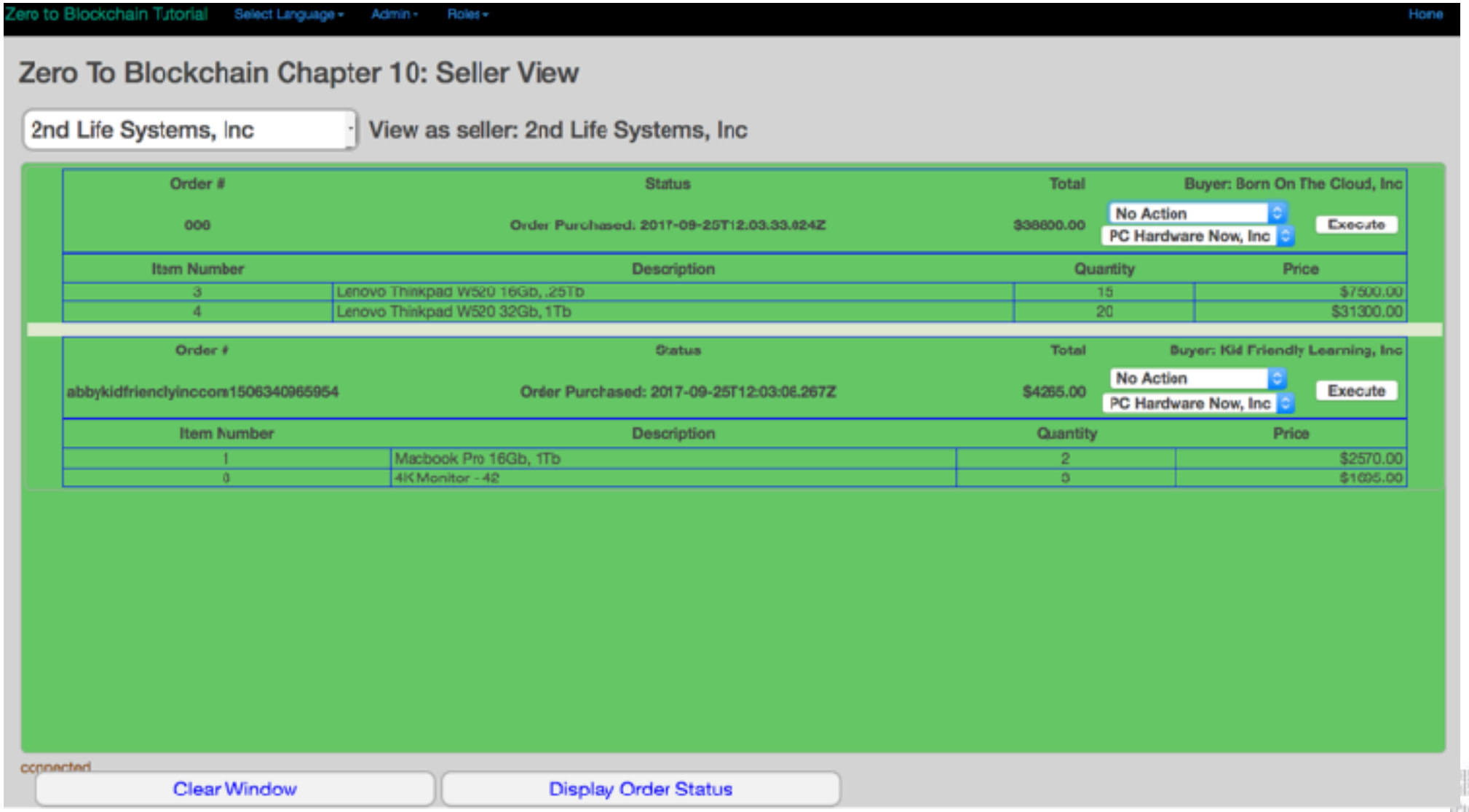
```
case 'Purchase':
console.log('Purchase entered');
updateOrder = factory.newTransaction(NS, 'Buy');
updateOrder.buyer = factory.newRelationship(NS, 'Buyer', order.buyer.$identifier);
updateOrder.seller = factory.newRelationship(NS, 'Seller', order.seller.$identifier);
break;
case 'Order From Supplier':
console.log('Order from Supplier entered for '+order.orderNumber+ ' inbound id: '+ _userID+' with c
updateOrder = factory.newTransaction(NS, 'OrderFromSupplier');
updateOrder.provider = factory.newRelationship(NS, 'Provider', req.body.provider);
updateOrder.seller = factory.newRelationship(NS, 'Seller', order.seller.$identifier);
break;
case 'Resolve':
console.log('Resolve entered');
updateOrder = factory.newTransaction(NS, 'Resolve');
updateOrder.buyer = factory.newRelationship(NS, 'Buyer', order.buyer.$identifier);
updateOrder.shipper = factory.newRelationship(NS, 'Shipper', order.shipper.$identifier);
updateOrder.provider = factory.newRelationship(NS, 'Provider', order.provider.$identifier);
updateOrder.seller = factory.newRelationship(NS, 'Seller', order.seller.$identifier);
updateOrder.financeCo = factory.newRelationship(NS, 'FinanceCo', financeCoID);
updateOrder.resolve = req.body.reason;
break;
case 'Dispute':
console.log('Dispute entered');
updateOrder = factory.newTransaction(NS, 'Dispute');
updateOrder.financeCo = factory.newRelationship(NS, 'FinanceCo', financeCoID);
updateOrder.buyer = factory.newRelationship(NS, 'Buyer', order.buyer.$identifier);
updateOrder.seller = factory.newRelationship(NS, 'Seller', order.seller.$identifier);
updateOrder.dispute = req.body.reason;
break;
case 'Authorize Payment':
console.log('Authorize Payment entered');
updateOrder = factory.newTransaction(NS, 'AuthorizePayment');
updateOrder.buyer = factory.newRelationship(NS, 'Buyer', order.buyer.$identifier);
updateOrder.financeCo = factory.newRelationship(NS, 'FinanceCo', financeCoID);
break;
case 'Cancel':
console.log('Cancel entered');
updateOrder = factory.newTransaction(NS, 'OrderCancel');
updateOrder.buyer = factory.newRelationship(NS, 'Buyer', order.buyer.$identifier);
updateOrder.seller = factory.newRelationship(NS, 'Seller', order.seller.$identifier);
break;
```

Seller capability:

- Order From Supplier
- Request Payment
- Resolve (a dispute)
- Refund (an order)

Note: Resolve has already been written for the buyer. We don't have to do anything to reuse it.

…Why?



The Plan: 30 minute Chapters with an hour or two of practice

Chapter 1:	What is Blockchain? Concept and Architecture overview
Chapter 2:	What's the story we're going to build
Chapter 2.1:	Architecture for the Story
Chapter 3:	Set up local HyperLedger V1 development environment
Chapter 4:	Build and test the network
Chapter 5:	Administration User Experience
Chapter 6:	Buyer Support and User Experience
Chapter 7:	Seller Support and User Experience
Chapter 8:	Shipper Support and User Experience
Chapter 9:	Provider Support and User Experience
Chapter 10:	Finance Company Support and User Experience
Chapter 11:	Combining for Demonstration
Chapter 12:	Events and Automating for Demonstration

Chapter 8: Shipper Support and User Experience

Learning Bluemix & Blockchain

Bob Dill, IBM Distinguished Engineer, CTO Global Technical Sales
David Smits, Senior Certified Architect, IBM Blockchain