BLOCKCHAINS

ARCHITECTURE, DESIGN AND USE CASES

SANDIP CHAKRABORTY
COMPUTER SCIENCE AND ENGINEERING,
IIT KHARAGPUR

PRAVEEN JAYACHANDRAN
IBM RESEARCH,
INDIA

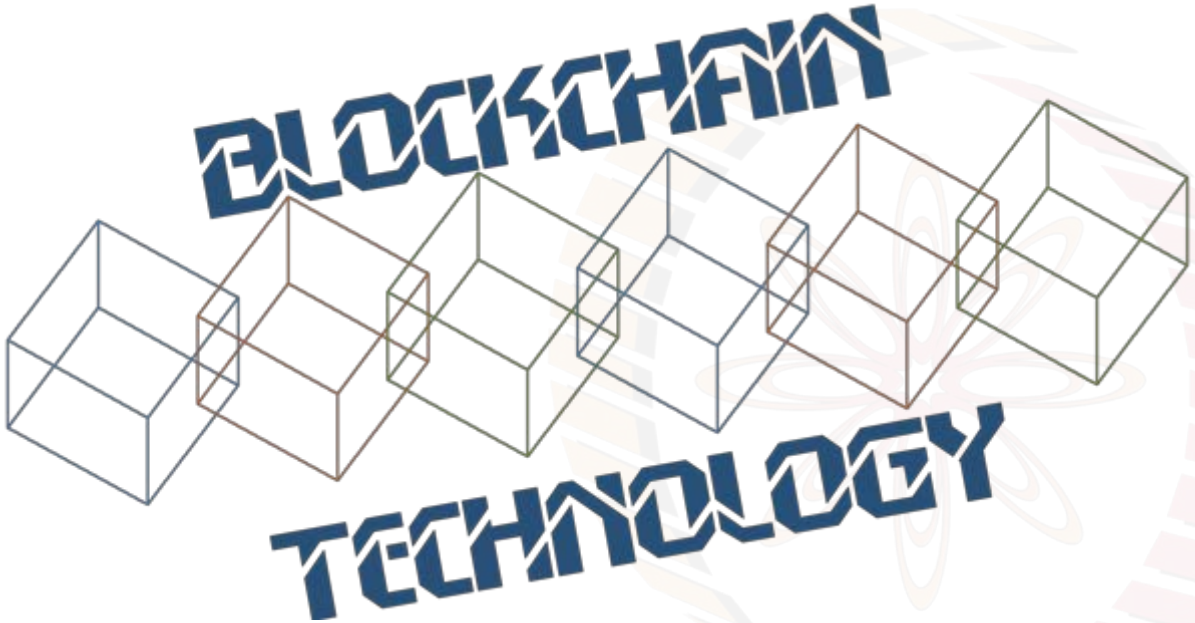


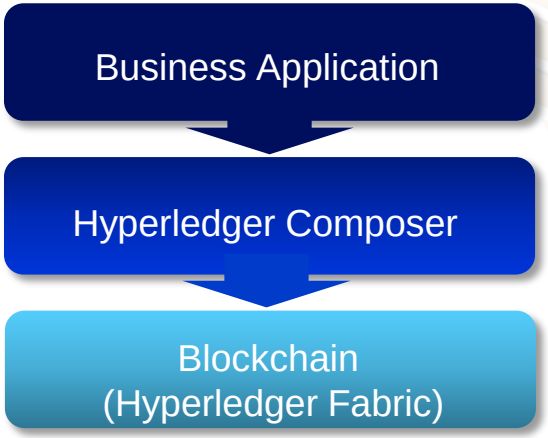
Image courtesy: http://beetfusion.com/

HYPERLEDGER COMPOSER - APPLICATION DEVELOPMENT

Hyperledger Composer: Accelerating Time to Value

- A suite of high level application abstractions for business networks to be built on top of Hyperledger Fabric
- Emphasis on **business-centric vocabulary** for quick solution creation – model in terms of assets, participants and transactions
- Reduce risk, and increase understanding and flexibility

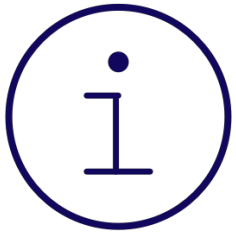
https://hyperledger.github.io/composer



- Features
  - Model your business networks, test and expose via APIs
  - Applications invoke transactions to interact with business network
  - Integrate existing systems of record
- **Fully open and part of Linux Foundation Hyperledger**

- Try it in your web browser now:
  http://composer-playground.mybluemix.net/

Goals of Hyperledger Composer



Increase understanding

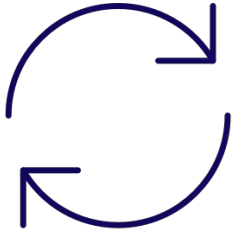
Bridges simply from business concepts to blockchain

Save time

Develop blockchain applications more quickly and cheaply

Reduce risk

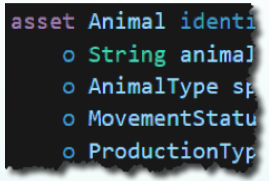
Well tested, efficient design conforms to best practice



Increase flexibility

Higher level abstraction makes it easier to iterate

Extensive, Familiar, Open Development Toolset



Data modelling

JavaScript
business logic

Web playground

```
composer-client
composer-admin
```

Client libraries

Editor support
(Atom, Visual Studio)

```
$ composer
```

CLI utilities

Code generation

Existing systems and
data

User Roles in a Blockchain Solution

- Network Service Provider
  - **Governs** the network: channels, membership etc.
  - A consortium of network members or designated authority

- Network Service Consumer
  - **Operates** a set of peers and certificate authorities on the network
  - Represents an organization on the business network

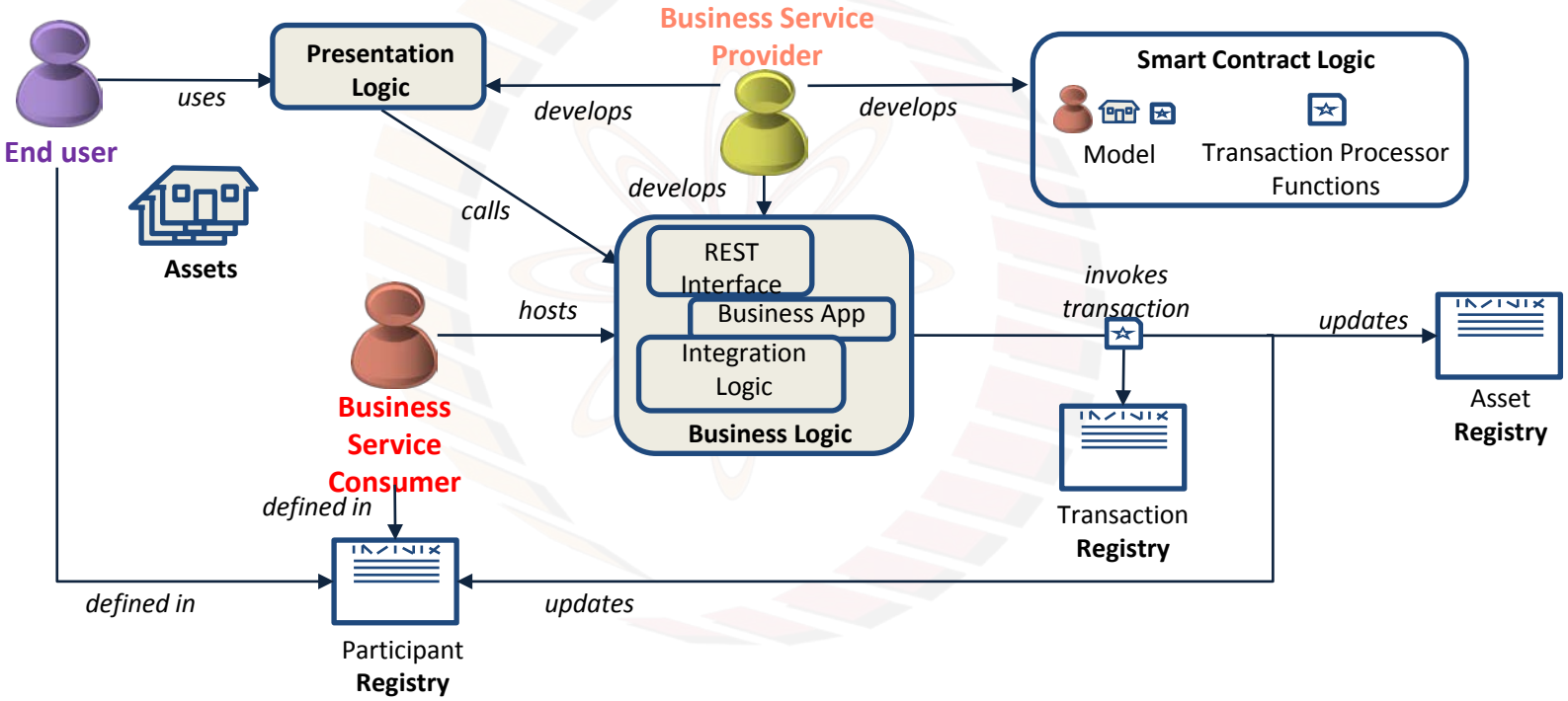
- Business Service Provider
  - **Develops** blockchain business applications
  - Includes transaction, app server, integration and presentation logic

- Business Service Consumer
  - Hosts application and integration logic which invokes blockchain transactions

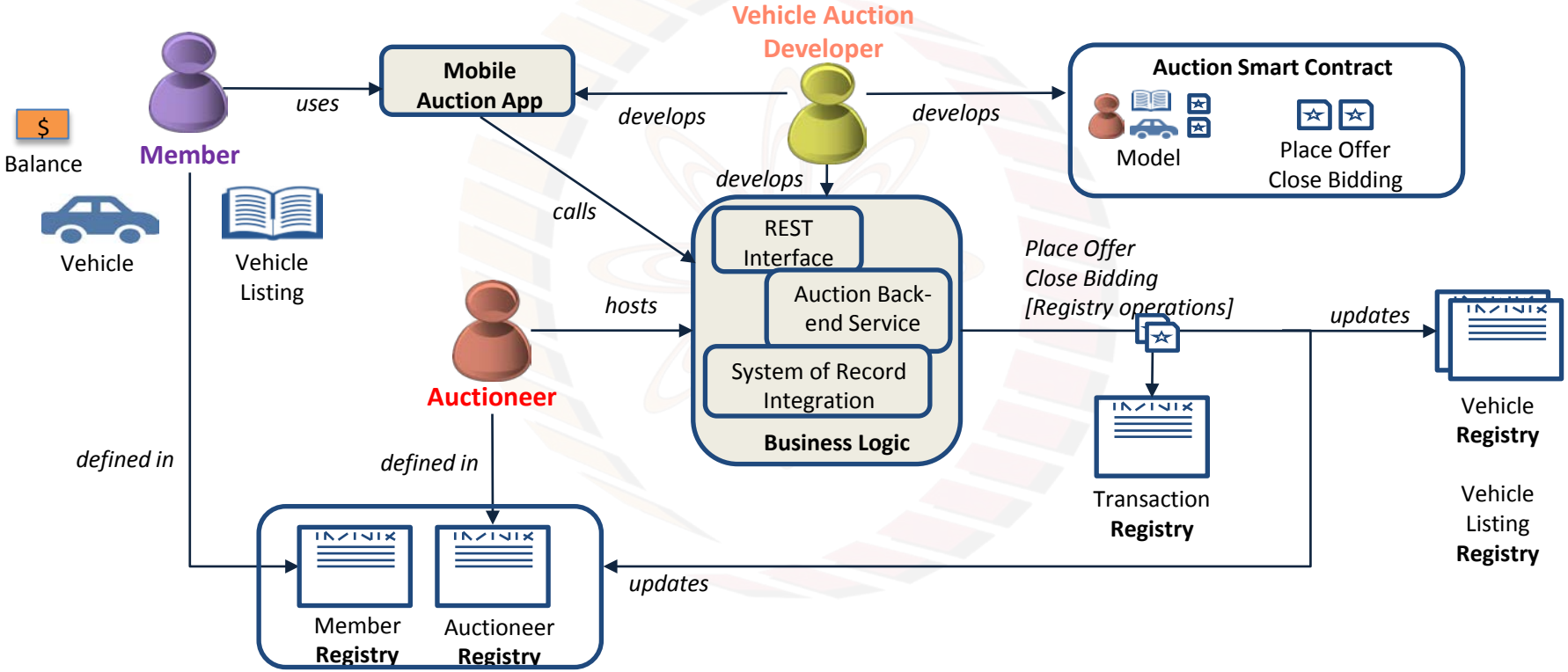
- End-user
  - Runs presentation logic e.g. on mobile device or dashboard

A single organization may play multiple roles!

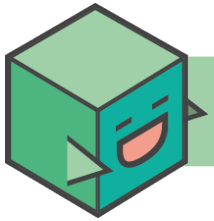
Key Concepts for the Business Service Provider



Example: Vehicle Auction Developer

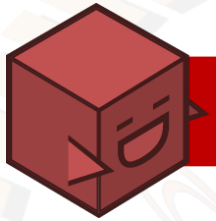


Business Service Provider develops three components



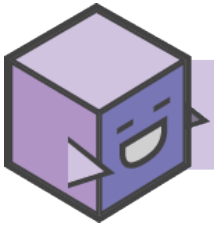
Smart Contracts

- Implements the logic deployed to the blockchain
  - **Models** describe assets, participants & transactions – expressive modeling language includes relationships and validation rules
  - **Transaction processors** provide the JavaScript implementation of transactions
  - **ACLs** define privacy rules
  - May also define events and registry queries



Business Logic

- **Services** that interact with the registries
  - Create, delete, update, query and invoke smart contracts
  - Implemented inside business applications, integration logic and REST services
- Hosted by the Business Application Consumer



Presentation Logic

- Provides the **front-end** for the end-user
  - May be several of these applications
- Interacts with business logic via standard interfaces (e.g. REST)
- Composer can generate the REST interface from model and a sample application

Key Development Concepts

- **Model files** describe the **assets, participants, and transactions** in a business network
  - Expressive modeling language includes relationships, arrays and validation rules
  - Data serialized as JSON, and is fully validated by Composer runtime (implemented as a chaincode in Fabric)
  - Model files can be shared and reused across business networks
- **Access control lists** define rules for sharing and privacy
  - Rules automatically enforced by Composer runtime
- **Transaction processors** implement additional business requirements
  - Standard javascript code executed on the Fabric network by Composer runtime
  - Each transaction processor function executed atomically on blockchain
- A **business network definition** is the set of the above for a given business
  - Has a name and version number

Assets, Participants and Transactions

Vehicle

Vehicle Listing

Member

Auctioneer

Place Offer
Close Bidding

```
asset Vehicle identified by vin {
  o String vin
  --> Member owner
}
```

```
asset VehicleListing identified by listingId {
  o String listingId
  o Double reservePrice
  o String description
  o ListingState state
  o Offer[] offers optional
  --> Vehicle vehicle
}
```

```
abstract participant User identified by email {
  o String email
  o String firstName
  o String lastName
}

participant Member extends User {
  o Double balance
}

participant Auctioneer extends User {
}
```

```
transaction Offer {
  o Double bidPrice
  --> VehicleListing listing
  --> Member member
}

transaction CloseBidding {
  --> VehicleListing listing
}
```

Transaction Processors

```
/**
 * Close the bidding
 * highest bid that is
 * @param {org.acme.ve
 * @transaction
 */
function closeBidding(
    var listing = clos
    if (listing.state
```

```
/**
 * Make an Offer for a VehicleListing
 * @param {org.acme.vehicle.auction.Offer} offer - the offer
 * @transaction
 */
function makeOffer(offer) {
    var listing = offer.listing;
    if (listing.state !== 'FOR_SALE') {
```

Access Control

```
rule EverybodyCanReadEverything {
    description: "Allow all participants read access to all resources"
    participant: "org.acme.sample.SampleParticipant"
    operation: READ
    resource: "org.acme.sample.*"
    action: ALLOW
}
```

```
rule OwnerHasFullAccessToTheirAssets {
    description: "Allow all participants full access to their assets"
    participant(p): "org.acme.sample.SampleParticipant"
    operation: ALL
    resource(r): "org.acme.sample.SampleAsset"
    condition: (r.owner.getIdentifier() === p.getIdentifier())
    action: ALLOW
}
```

```
rule SystemACL {
  description:  "System ACL to permit all access"
  participant: "org.hyperledger.composer.system.Participant"
  operation: ALL
  resource: "org.hyperledger.composer.system.**"
  action: ALLOW
}
```

- It is possible to restrict which resources can be read and modified by which participants
  - Rules are defined in an .acl file and deployed with the rest of the model
  - Transaction processors can also look up the current user and implement rules programmatically
- ACL rules can be simple (e.g. everybody can read all resources) or more complex (e.g. only the owner of an asset can do everything to it)
- Application supplies credentials (userid/secret) of the participant when connecting to the Fabric network
  - This also applies to Playground!

Events and Queries

- Events allow applications to take action when a transaction occurs
  - Events are **defined** in models
  - Events are **emitted** by transaction processor scripts
  - Events are **caught** by business applications
- Caught events include transaction ID and other relevant information

```
event SampleEvent {
  --> SampleAsset asset
  o String oldValue
  o String newValue
}
```

```
// Emit an event for the modified asset.
var event = getFactory().newEvent('org.acme.sample', 'SampleEvent');
event.asset = tx.asset;
event.oldValue = oldValue;
event.newValue = tx.newValue;
emit(event);
```

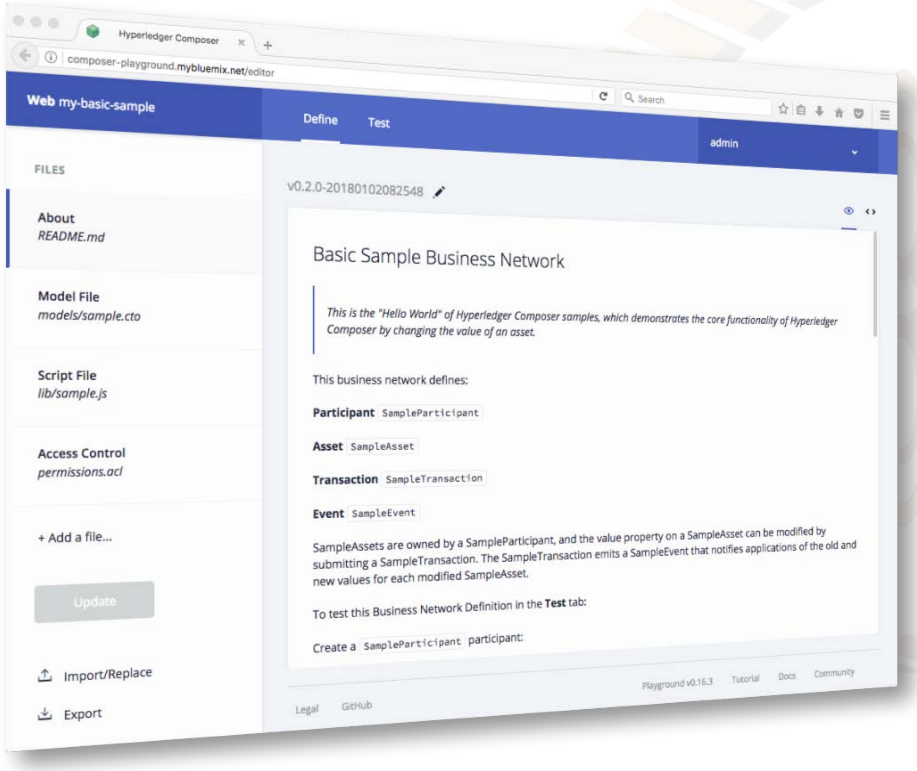
```
businessNetworkConnection.on('SampleEvent', (event) => {
    console.log(event);
}
```

- Queries allow applications to perform complex registry searches
  - They can be statically defined in a separate .qry file or generated dynamically by the application
  - They are invoked in the application using *buildQuery*() or *query*()
  - Queries require the blockchain to be backed by CouchDB

```
query selectCommoditiesWithHighQuantity {
  description: "Select commodities based on quantity"
  statement:
      SELECT org.acme.trading.Commodity
          WHERE (quantity > 60)
}
```

```
return query('selectCommoditiesWithHighQuantity', {})
```

Smart Contract Development: Composer Playground



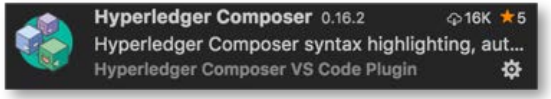
- Web tool for defining and testing Hyperledger Composer models and scripts
- Designed for the application developer
  - Define assets, participants and transactions
  - Implement transaction processor scripts
  - Test by populating registries and invoking transactions
- Deploy to instances of Hyperledger Fabric V1, or simulate completely within browser
- Install on your machine or run online at http://composer-playground.mybluemix.net

General Purpose Development: Visual Studio Tool

- Composer extension available for this popular tool

- Features to aid rapid Composer development
  - Edit all Composer file types with full syntax highlighting, code completion
  - Validation support for models, queries and ACLs
  - Inline error reporting
  - Snippets (press Ctrl+Space for code suggestions)
  - Generate UML diagrams from models

- Install directly from Code Marketplace

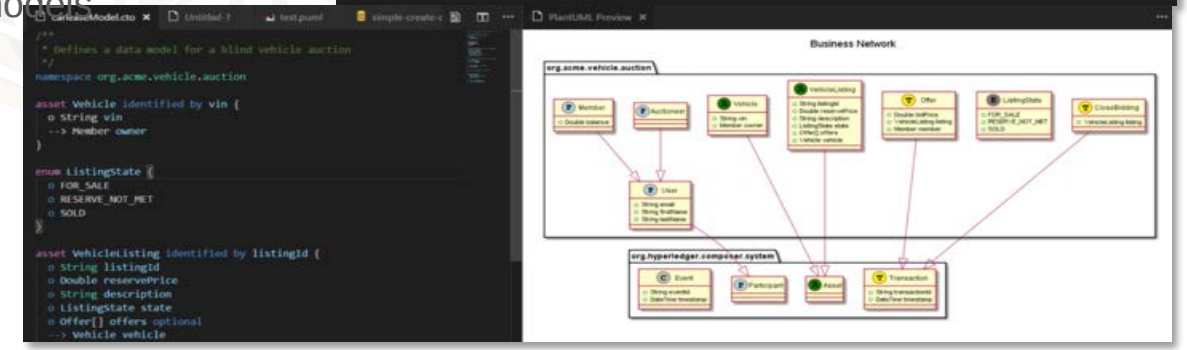



```
namespace org.acme.vehicle.lifecycle

import composer.base.Person
import composer.business.Business

participant PrivateOwner identified by email extends Person {
  o String email
}
```

```
[Composer] IllegalModelException: Could not find super type Pea
rson
participant PrivateOwner identified by email extends Pearson {
  o String email
}
```

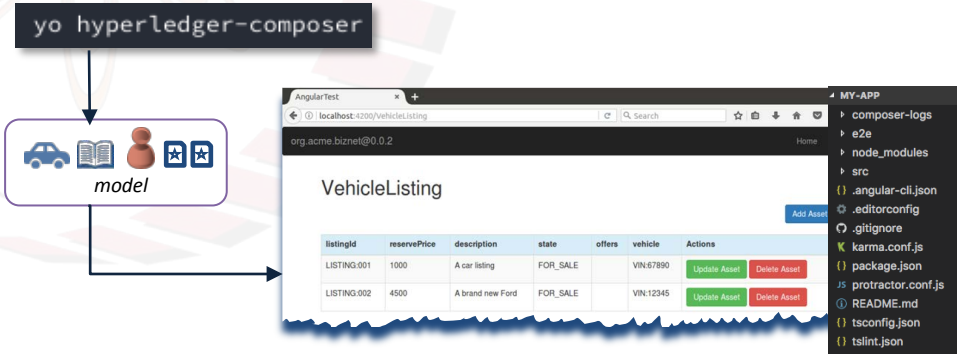


Creating the Business and End-User Applications

- JavaScript business applications *require*() the NPM “composer-client” module
  - This provides the API to access assets, participants and transactions
  - RESTful API (via Loopback) can also be generated… see later

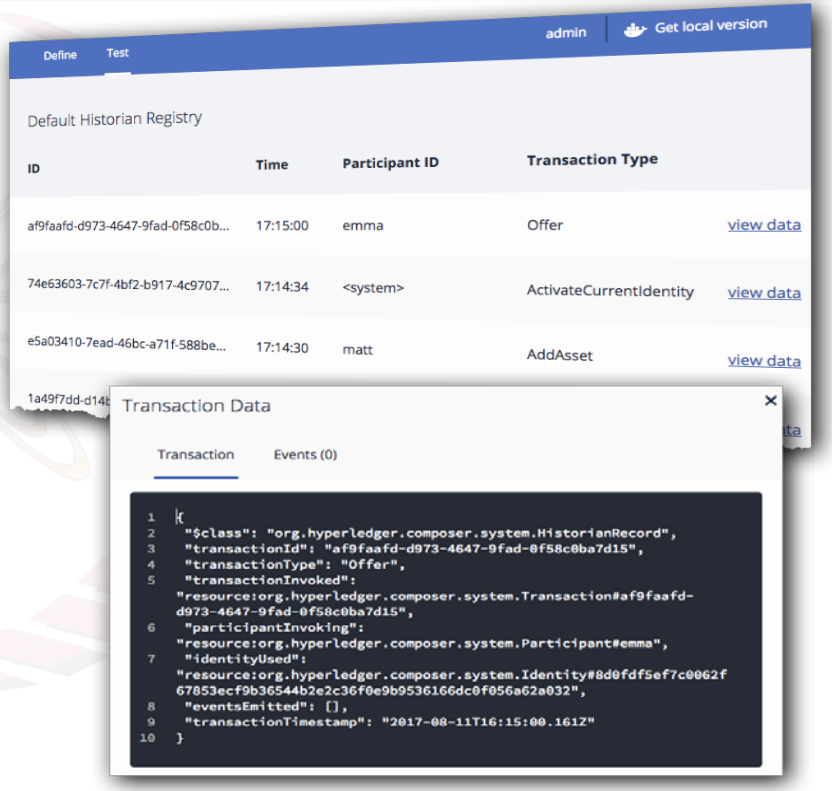
- Command-line tool available to generate end-user command-line or Angular2 applications from model
  - Also helps with the generation of unit tests to help ensure quality code

```
const BusinessNetworkConnection = require('composer-client')
                                    .BusinessNetworkConnection;
this.bizNetworkConnection = new BusinessNetworkConnection();
this.titlesRegistry = this.bizNetworkConnection.getAssetRegistry
                        ('net.biz.digitalPropertyNetwork.LandTitle')
let landTitle1 = factory.newResource
    ('net.biz.digitalPropertyNetwork', 'LandTitle', 'LID:1148');
landTitle1.owner = ownerRelation;
landTitle1.information = 'A nice house in the country';
this.titlesRegistry.add(landTitle1);
```



Debugging

- Playground Historian allows you to view all transactions
  - See what occurred and when
- Diagnostics framework allows for application level trace
  - Uses the *Winston* Node.js logging framework
  - Application logging using DEBUG env var
  - Composer Logs sent to stdout and ./logs/trace_<processid>.trc
- Fabric chaincode tracing also possible
- More information online: https://hyperledger.github.io/composer/problems/diagnostics.html



Fun Reading

- What is Hyperledger Composer (6 mins): https://www.youtube.com/watch?v=PvrLJTGfje0
- Introduction to Hyperledger Composer (58 minutes): https://www.youtube.com/watch?v=fdFUsrsv5iw
- The Accord Project, Smart Legal Contracts: http://www.accordproject.org/


thank you!