# Order Something!项目文档

---

- 项目成员：
- 区块链&智能合约编写：肖阳、李季航
- 前端设计：杨铭、孙尧、熊路阳
- ppt制作：白致宁、李季航
- ppt讲解：杨铭
- 项目日期：2019年7月19日
- 项目网址：https://github.com/WArushrush/hyperledger-blockchain

## PART ONE ——项目设计

### 一基础区块网络&智能合约

- 业务场景：客户利用网络平台进行商品交易
  - 1、用户对平台下订单并计算出用户距离。根据用户距离和商品价格，平台存储订单信息为未完成状态，暂存订金。商家收到订单，准备发货。
  - 2、商家确认订单后将商品送出。用户准备收货，平台准备确认订单完成。
  - 3、用户确认收货后，平台记录订单已完成并在订金中扣除运费，向商家支付相应金额。

- model.cto 组成：
  - **asset**
    - Customer

```
asset Customer identified by ID {
  o String ID
  o Double deposit
  o String[] commodities//客户拥有的商品
  o Integer index//客户拥有的商品数
  --> Person owner
}
```

    - Merchant

```
asset Merchant identified by ID {
  o String ID
  o String Name //商家姓名
  o Double deposit
  o Double x //商家坐标x
  o Double y //商家坐标y
  o Double[] commodities//id索引商品价格
  o String[] comname//货物名称
  o Integer index//指待发货
```

```
  --> Customer[] cust//顾客名录
 --> Platform[] plat//平台名录
 o Integer[] commidToDeliver//相应要发的商品id
  --> Person owner
}
```

- Platform

```
asset Platform identified by ID{
 o String ID
 o String Name
 o Double deposit
 o Double[] commidToPay//待付款值
 o Integer index//指待付款
 o Integer Total//执行的订单总数
 o Integer Finish//完成的订单总数
-->Merchant[] domer//记录对应执行订单的用户
-->Customer[] docus//记录对应执行订单的商家
 o String[] state//记录对应执行订单的的状态
--> Merchant[] merc//相应要付款的商家
}
```

**- participant-Person**

```
participant Person identified by ID {
o String ID
o String Name
}
```

**- transaction**

- Order

```
transaction Order{//下单：客户指定num，客户+num存进mc
--> Merchant mc
 --> Customer ct
 --> Platform pt
 o Integer num
 o Double x//客户目前x坐标
 o Double y//客户目前y坐标
}
```

- Deliver

```
transaction Deliver {
//送货：mc从index-1位置得到发货num:=id+减对应客户；ct在index位置存活+待付款商家，index+1
 --> Merchant mc
}
```

- Pay

```
transaction Pay {
 //付款：ct从index-1位置得到付款商家+付款商品号num:=id；用num在mt商品数组中索引bill；ct.deposit-bill&&mc.deposit+bill
--> Platform pt
}
```

- query_customer

```
transaction query_customer{
o String id
}
```

- logistics

```
transaction logistics{
--> Merchant from
```

```
    --> Customer to
    o Integer num
    }
```

- script.js 组成
  - **function Order**

```
/**
 * Sample transaction processor function.
 * @param {org.example.empty.Order} tx The sample transaction
instance.
 * @transaction
 */
async function Order(tx){
    tx.mc.commidToDeliver[tx.mc.index] = tx.num;//对应商家添加订
单，订单号对应到商品
    tx.mc.cust[tx.mc.index] = tx.ct;     //订单号对应到客户
    tx.mc.plat[tx.mc.index] = tx.pt;     //订单号对应到平台
    tx.mc.index += 1;                    //对应商家订单总数加一
    tx.ct.deposit -= tx.mc.commodities[tx.num]
+Math.sqrt((tx.mc.x-tx.x)*(tx.mc.x-tx.x)+(tx.mc.y-
tx.y)*(tx.mc.y-tx.y));//对应客户付款
    tx.pt.deposit += tx.mc.commodities[tx.num]
+Math.sqrt((tx.mc.x-tx.x)*(tx.mc.x-tx.x)+(tx.mc.y-
tx.y)*(tx.mc.y-tx.y));//对应平台收款
    tx.pt.domer[tx.pt.Total]=tx.mc;
    tx.pt.docus[tx.pt.Total]=tx.ct;
    tx.pt.state[tx.pt.Total]="Unfinished";
    tx.pt.Total +=1;
    const assetRegistry1 = await
getAssetRegistry('org.example.empty.Merchant');//等待更新
    await assetRegistry1.update(tx.mc);
```

```
    const assetRegistry2 = await
getAssetRegistry('org.example.empty.Platform');
    await assetRegistry2.update(tx.pt);
    const assetRegistry3 = await
getAssetRegistry('org.example.empty.Customer');
    await assetRegistry3.update(tx.ct);
}
```

**- function Deliver**

```
/**
 * Sample transaction processor function.
 * @param {org.example.empty.Deliver} tx The sample
transaction instance.
 * @transaction
 */
async function Deliver(tx){
    tx.mc.index -= 1;//商家对应的商品数减1
    let num=tx.mc.commidToDeliver[0];//取出第一个要送的商品，对应顾
客和对应平台
    let ct = tx.mc.cust[0];
    let pt = tx.mc.plat[0];
    //所有订单号对应的商品，顾客和平台都前移一个
    //删除最后一个订单信息
    tx.mc.commidToDeliver.shift();
    tx.mc.cust.shift();
    tx.mc.plat.shift();
    //对应平台要付的货款加1，对应到对应的货款和商家
    pt.commidToPay[pt.index] = tx.mc.commodities[num];
    pt.merc[pt.index] = tx.mc;
    pt.index += 1;
    //对应客户拥有的货物
    ct.commodities[ct.index] = tx.mc.comname[num];
    ct.index +=1;
```

```
    //等待更新
    const assetRegistry1 = await
getAssetRegistry('org.example.empty.Merchant');
    await assetRegistry1.update(tx.mc);
    const assetRegistry2 = await
getAssetRegistry('org.example.empty.Platform');
    await assetRegistry2.update(pt);
    const assetRegistry3 = await
getAssetRegistry('org.example.empty.Customer');
    await assetRegistry3.update(ct);
  }
```

**- function Pay**

```
/**
 * Sample transaction processor function.
 * @param {org.example.empty.Pay} tx The sample transaction
instance.
 * @transaction
 */
async function Pay(tx){
     tx.pt.deposit-=tx.pt.commidToPay[0];//平台支付货款
    tx.pt.merc[0].deposit+=tx.pt.commidToPay[0];//商家收款
    tx.pt.state[tx.pt.Finish]="Finished";
    tx.pt.Finish +=1;
    const assetRegistry1 = await
getAssetRegistry('org.example.empty.Merchant');
    await assetRegistry1.update(tx.pt.merc[tx.pt.index-1]);
    tx.pt.merc.shift();
    tx.pt.commidToPay.shift();
    tx.pt.index-=1;//平台待支付价款减一
    const assetRegistry2 = await
getAssetRegistry('org.example.empty.Platform');
    await assetRegistry2.update(tx.pt);
```

```
}
```

- **function query_customer**

```
/**
 * Sample transaction processor function.
 * @param {org.example.empty.query_customer} tx The sample
transaction instance.
 * @transaction
 */
  async function query_customer(tx){
   let assetRegistry1 = await
getAssetRegistry('org.example.empty.Customer');
   let resource = await assetRegistry1.getAll();
   let l = resource.length;
   var money=0;
   for(let i=0; i<l;i++){
      if(resource[i].ID==tx.ID){
          money=resource[i].deposit;
      }
   }
    console.log(money);
    return money;
  }
```

- **function logistics**

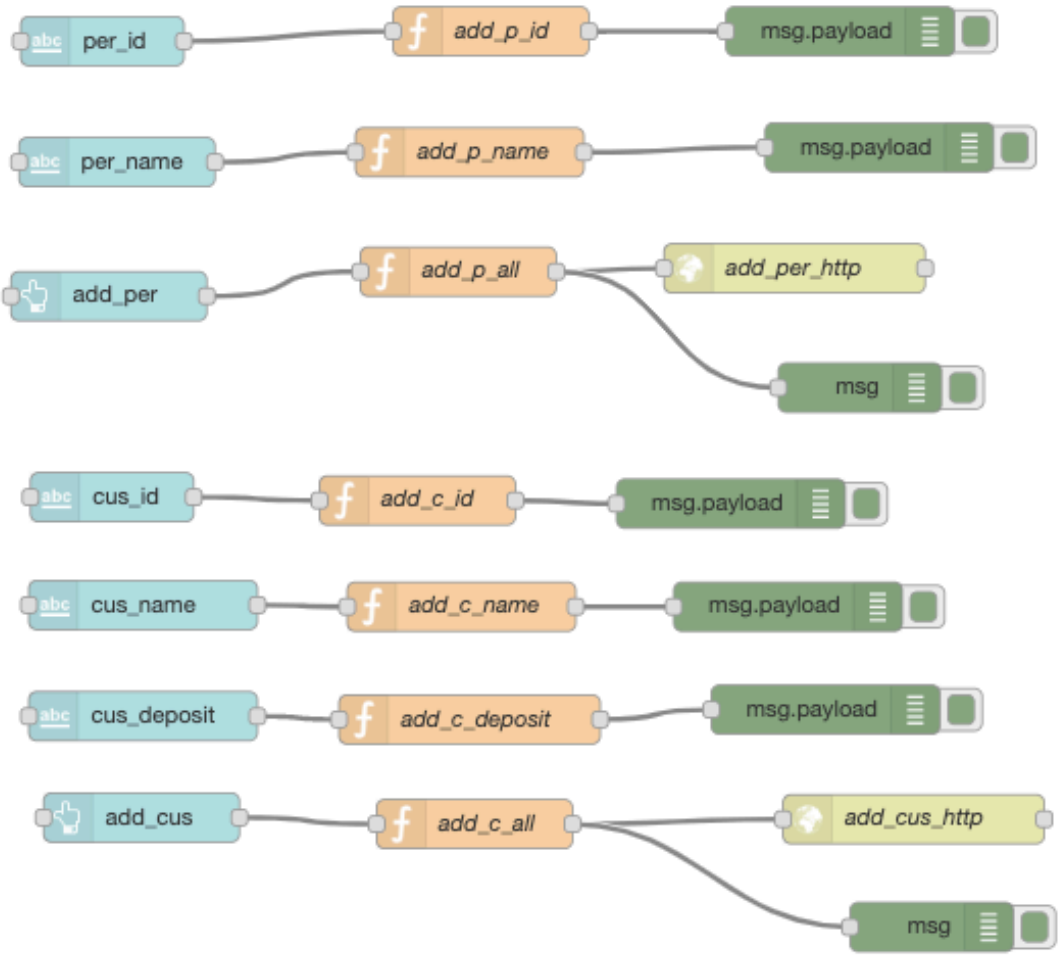
```
/**
 * Sample transaction processor function.
 * @param {org.example.empty.logistics} tx The sample
transaction instance.
 * @transaction
 */
async function logistics(tx){
  console.log("commodity transfer successful");
```

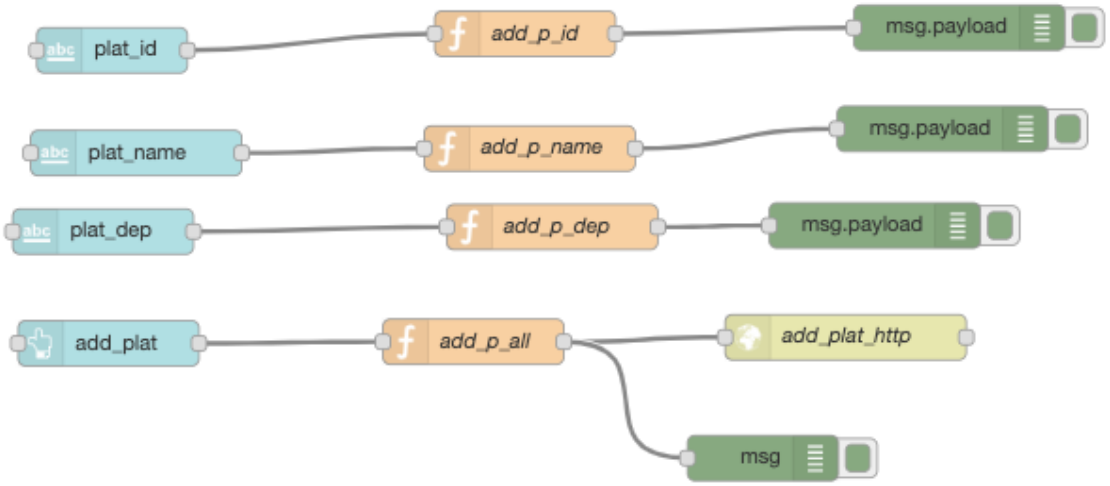
}

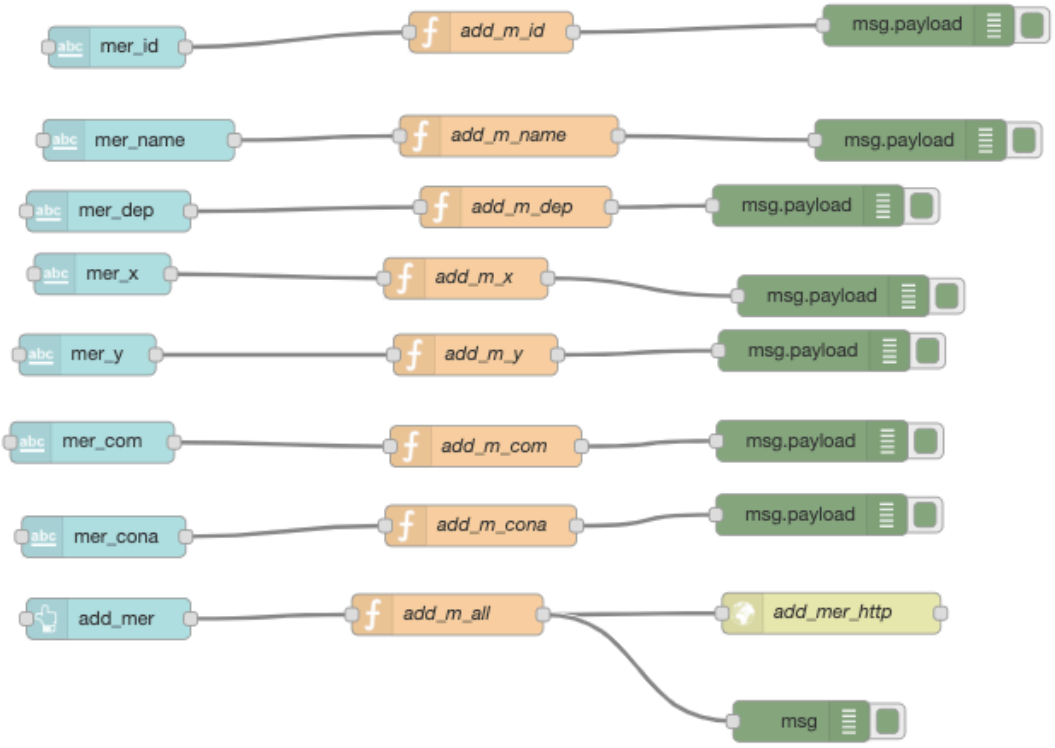
## —Node-red前端设计

- 流程图

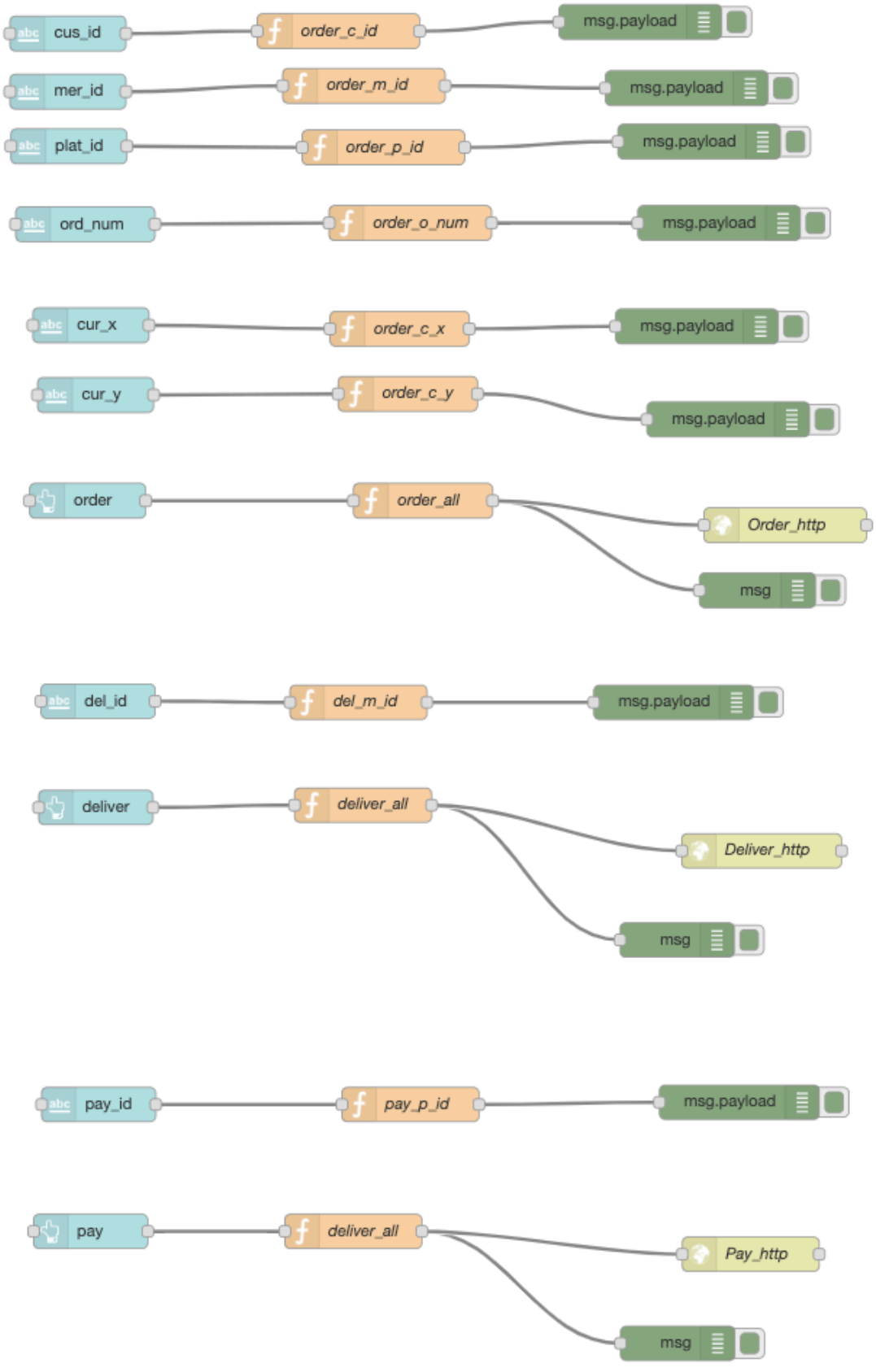
  — 添加数据



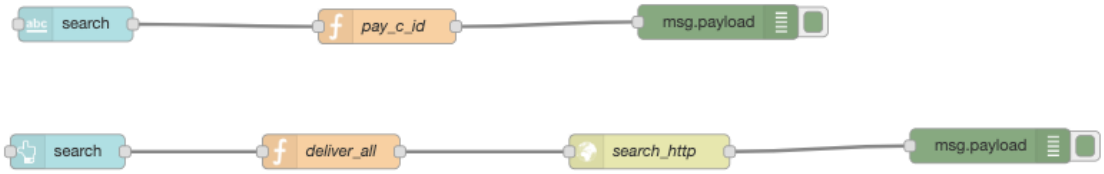

plat_id
add_p_id
msg.payload
plat_name
add_p_name
msg.payload
plat_dep
add_p_dep
msg.payload
add_plat
add_p_all
add_plat_http
msg


mer_id
add_m_id
msg.payload
mer_name
add_m_name
msg.payload
mer_dep
add_m_dep
msg.payload
mer_x
add_m_x
msg.payload
mer_y
add_m_y
msg.payload
mer_com
add_m_com
msg.payload
mer_cona
add_m_cona
msg.payload
add_mer
add_m_all
add_mer_http
msg

– 交易实现

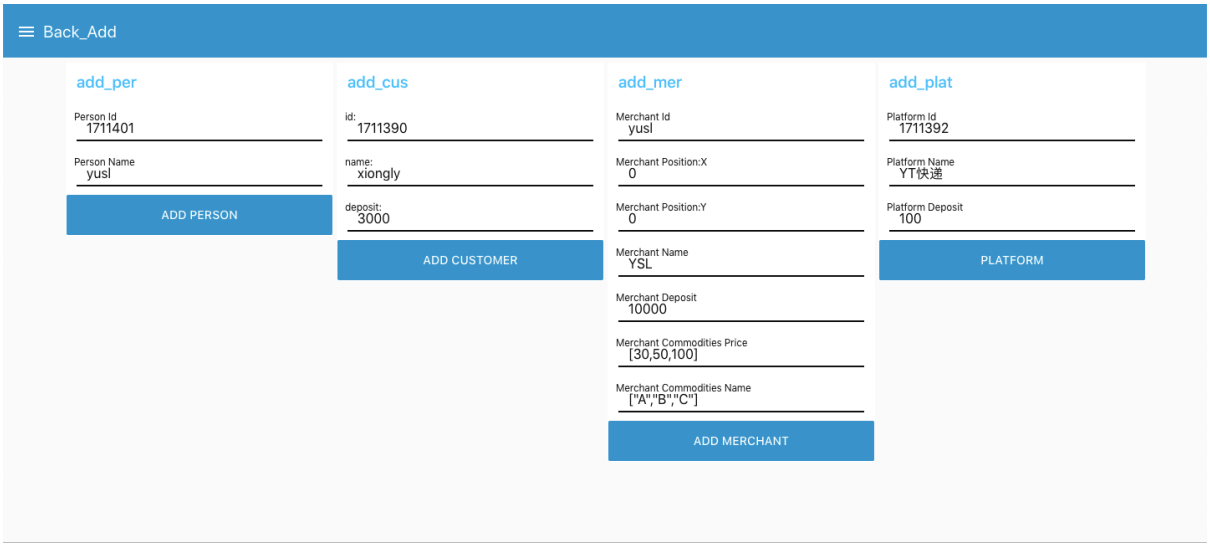

cus_id
order_c_id
msg.payload
mer_id
order_m_id
msg.payload
plat_id
order_p_id
msg.payload
ord_num
order_o_num
msg.payload
cur_x
order_c_x
msg.payload
cur_y
order_c_y
msg.payload
order
order_all
Order_http
msg
del_id
del_m_id
msg.payload
deliver
deliver_all
Deliver_http
msg
pay_id
pay_p_id
msg.payload
pay
deliver_all
Pay_http
msg

- 查找用户

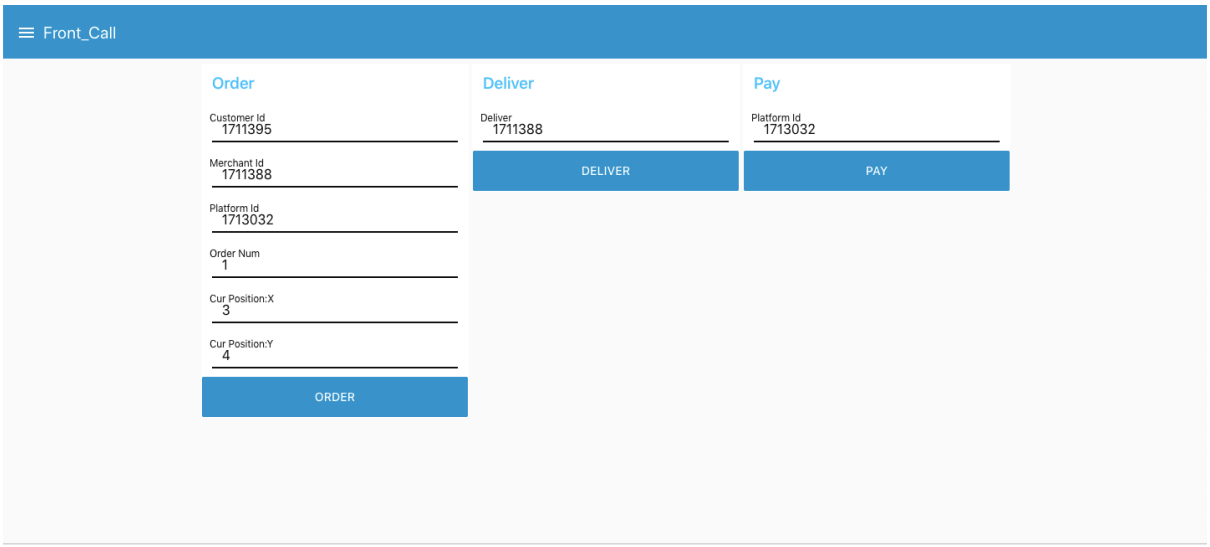



- 效果图

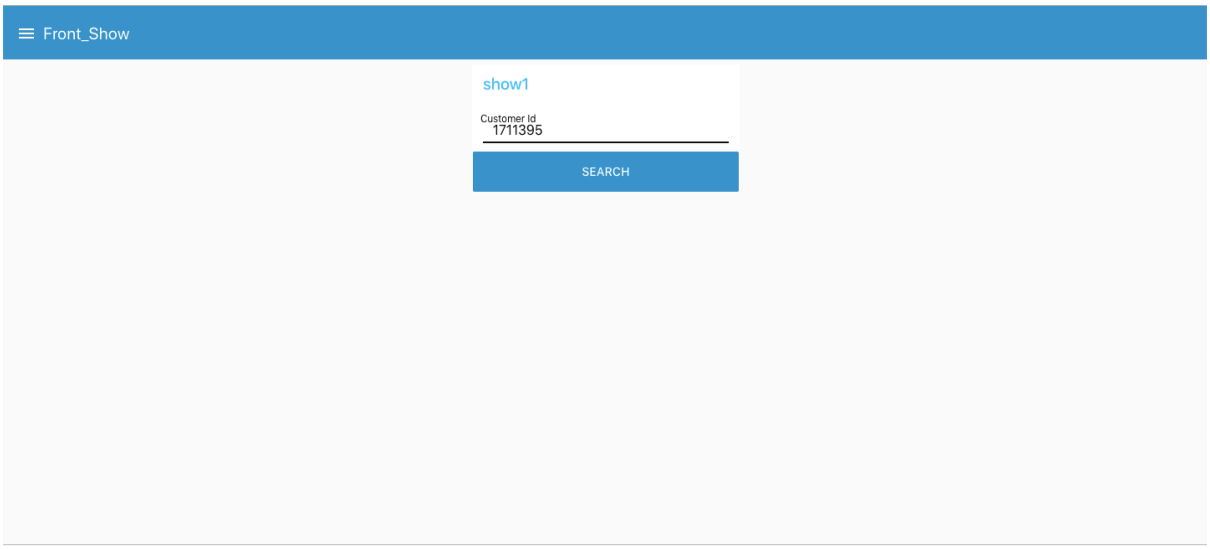
- 添加用户



– 交易实现


Front_Call
Order
Customer Id
1711395
Merchant Id
1711388
Platform Id
1713032
Order Num
1
Cur Position:X
3
Cur Position:Y
4
ORDER
Deliver
Deliver
1711388
DELIVER
Pay
Platform Id
1713032
PAY

– 查询用户


Front_Show
show1
Customer Id
1711395
SEARCH

## PART TWO ——Readme

```
# Order Something!

基础网络定义:

**Participant**
`Person`

**Asset**
`Customer`
`Merchant`

**Transaction**
`Order`
`Deliver`
`Pay`
`logistics`

To test this Business Network Definition in the **Test** tab:

Create two `Person` participants:

```
{
    "$class": "org.example.empty.Person",
    "ID": "1",
    "Name": "Toby"
}
{
    "$class": "org.example.empty.Person",
    "ID": "2",
    "Name": "Bob"
}
```

Create a`Customer`asset:

```
{
    "$class": "org.example.empty.Customer",
    "ID": "1",
    "deposit": 0,
    "commodities": [],
    "index": 0,
    "owner": "resource:org.example.empty.Person#1"
}
```
```

```

Create a`Merchant`asset:

```
{
    "$class": "org.example.empty.Merchant",
    "ID": "2",
    "owner": "resource:org.example.empty.Person#2",
    "deposit": "original value",
    "x": 0,
    "y": 0,
    "commodities": [],
    "comname": [],
    "index":0,
    "cust": [],
    "plat": [],
    "commidToDeliver": [],
    "owner": "resource:org.example.empty.Person#2"
}
```

Create a`Platform`asset:

```
{
    "$class": "org.example.empty.Platform",
    "ID": "3822",
    "Name": "",
    "deposit": 0,
    "commidToPay": [],
    "index": 0,
    "Total": 0,
    "Finish": 0,
    "domer": [],
    "docus": [],
    "state": [],
    "merc": []
}
```

Submit a `Order` transaction:

```
{
    "$class": "org.example.empty.Order",
    "mc": "resource:org.example.empty.Merchant#2",
    "ct": "resource:org.example.empty.Customer#1",
```
```

```
    "pt": "resource:org.example.empty.Platform#3",
    "num": 4,
    "x": 1,
    "y": 1

}
```

Submit a `Deliver` transaction:

```
{
    "$class": "org.example.empty.Deliver",
    "mc": "resource:org.example.empty.Merchant#ID:2",

}
```

Submit a `Pay` transaction:

```
{
    "$class": "org.example.empty.Pay",
    "pt": "resource:org.example.empty.Platform#3"

}
```

Submit a `logistics` transaction:

```
{
    "$class": "org.example.empty.logistics",
    "from": "resource:org.example.empty.Merchant#2",
    "to": "resource:org.example.empty.Customer#1",
    "num": 4
}
```

Submit a `query_customer` transaction:

```
{
    "$class": "org.example.empty.query_customer",
    "id": ""
}
```

在“Order”阶段，首先系统根据客户的坐标确定运费。之后客户可对我们的外卖平台提交订单（指定一个平台、商品编号、商家）。商家将用户记录在cust数组中，将订单存入对应平台数组plat中，将对应商品编号存入commidToDeliver，并将“指针”变量“index”+1。将通过commodities数组用商品编号索引出商品价值，平台根据商品价格和运费对客户进行收款，此时，客户的存款减去相应的金额，平台存款增加。同时，平台将此时相关联的客户、商家计入平台维护的数组docus和domer中，将该单状态存储为“unfinished”状态。平台正在执行的订单总数“Total”+1。

"Deliver"阶段，在商家asset的定义中，我们使用“index”变量记录从0位置到index位置为商家此刻还没有进行发货的订单。当商家发起“Deliver”操作时，系统自动为商家根据cust数组中记录的客户顺序和commidToDeliver中记录的商品顺序发出缓冲列表中的第一单，即最早进行的订单。后续的客户和商品依次向数组头移动一位，后续收到的订单被记录在此时的“index”之后。平台此时在merc数组和commidToPay记录待付款的商家和对应商家编号。存储对应客户所拥有的商品的commodities数组从商家comname数组中用商品id索引出商品名称，记录在commodities中。

在对商家付款的“Pay”阶段，平台首先从缓冲表commidToPay[0]和merc[0]位置取出当前要支付的商品编号和相应的商家。平台的存款减去商品的金额，为对应商家存款t增加金额。根据“index”将两个数组中后续的商品编号和商家前移，后续收到的订单被记录在此时的“index”之后。付款完成后，该订单的状态在平台中被更新为“finished”。平台已完成的订单总数“Finished”+1。

“logistics”可为我们提示一系列交易已成功进行。

“query_customer”可以实现查看指定客户实时的存款金额。

进行完上述一系列操作，客户commodities中新增商品编号为4的商品名称，客户存款中扣除运费和该商品的价格。平台z中记录了此次进行交易的双方Customer1&Merchant2，Total+1，Finished+1。平台deposit+运费，商家deposit+商品价格。