Source file src/chaincode/chaincode_ad/chaincode_ad_test.go

Documentation: **chaincode/chaincode_ad**

```
 1  package main
 2
 3  import (
 4          "fmt"
 5          "testing"
 6
 7          "github.com/hyperledger/fabric/core/chaincode/shim"
 8  )
 9
10  func checkInit(t *testing.T, stub *shim.MockStub, args [][]byte) {
11          res := stub.MockInit("1", args)
12          if res.Status != shim.OK {
13                  fmt.Println("Init failed", string(res.Message))
14                  t.Fail()
15          }
16  }
17
18  func checkInvoke(t *testing.T, stub *shim.MockStub, args [][]byte) {
19          res := stub.MockInvoke("1", args)
20          if res.Status != shim.OK {
```

```
    21                  fmt.Println("Invoke", args, "failed", string(res.Message))
    22                  t.Fail()
    23          }
    24  }
    25
    26  func checkInvokeFail(t *testing.T, stub *shim.MockStub, args [][]byte) {
    27          res := stub.MockInvoke("1", args)
    28          if res.Status == shim.OK {
    29                  fmt.Println("Invoke", args, "should fail but did not", string(res.Payload))
    30                  t.Fail()
    31          }
    32  }
    33
    34  func checkInvokeResponse(t *testing.T, stub *shim.MockStub, args [][]byte, expectedPayload
string) {
    35          res := stub.MockInvoke("1", args)
    36          if res.Status != shim.OK {
    37                  fmt.Println("Invoke", args, "failed", string(res.Message))
    38                  t.Fail()
    39          }
    40          if string(res.Payload) != expectedPayload {
    41                  fmt.Println("Expected payload:", expectedPayload)
    42                  fmt.Println("Instead got this:", string(res.Payload))
    43                  t.Fail()
    44          }
    45  }
    46
    47  func checkInvokeResponseFail(t *testing.T, stub *shim.MockStub, args [][]byte, expectedMessage
string) {
    48          res := stub.MockInvoke("1", args)
    49          if res.Status == shim.OK {
```

```
50                  fmt.Println("Invoke", args, "should fail")
51                  fmt.Println("Instead got payload:", string(res.Payload))
52                  t.Fail()
53          }
54          if res.Message != expectedMessage {
55                  fmt.Println("Expected message:", expectedMessage)
56                  fmt.Println("Instead got this:", res.Message)
57                  t.Fail()
58          }
59  }
60
61  func Test_Init(t *testing.T) {
62          cc := new(Chaincode)
63          stub := shim.NewMockStub("init_test", cc)
64
65          // Init should always success
66          checkInit(t, stub, [][]byte{[]byte("1")})
67  }
68
69  func Test_InvokeFail(t *testing.T) {
70          cc := new(Chaincode)
71          stub := shim.NewMockStub("invoke_fail_test", cc)
72          args := [][]byte{[]byte("NoFunction"), []byte("test")}
73          expectedMessage := "Received unknown function invocation"
74          checkInvokeResponseFail(t, stub, args, expectedMessage)
75  }
76
77  func Test_createDataEntryAd(t *testing.T) {
78          cc := new(Chaincode)
79          stub := shim.NewMockStub("init_test", cc)
80
```

```
81          // Init
82          checkInit(t, stub, [][]byte{[]byte("20")})
83          // create test data entry ad
84          args := [][]byte{[]byte("createDataEntryAd"),
85                  []byte("1"), []byte("test_data"), []byte("???"), []byte("Unit"),
86                  []byte("20181212152030"), []byte("pub_name"), []byte("1"), []byte("2")}
87          checkInvokeResponse(t, stub, args, "")
88
89          // Check if it is in the state
90          args = [][]byte{[]byte("getDataAdByIDAndTime"), []byte("1"), []byte("20181212152030")}
91          expectedPayload := "{\"RecordType\":\"DATA_ENTRY_AD\",\"DataEntryID\":\"1\"" +
92                  ",\"Description\":\"test_data\",\"Value\":\"???\",\"Unit\":\"Unit\"," +
93                  "\"CreationTime\":20181212152030,\"Publisher\":\"pub_name\"," +
94                  "\"Price\":1,\"AccountNo\":\"2\"}"
95          checkInvokeResponse(t, stub, args, expectedPayload)
96
97          // create test data entry ad that has the same ID and creationTime Shoult fail
98          args = [][]byte{[]byte("createDataEntryAd"),
99                  []byte("1"), []byte("test_data"), []byte("50"), []byte("Unit"),
100                 []byte("20181212152030"), []byte("pub_name"), []byte("10"), []byte("2")}
101         expectedMessage := "This data entry already exists: 1~20181212152030"
102         checkInvokeResponseFail(t, stub, args, expectedMessage)
103
104         // It should fail to createDataEntryAd that have one empty arg
105         args = [][]byte{[]byte("createDataEntryAd"),
106                 []byte(""), []byte("test_data"), []byte("???"), []byte("Unit"),
107                 []byte("20181212152030"), []byte("pub_name"), []byte("10"), []byte("2")}
108         // it should not save to the state and it should fail
109         expectedMessage = "Argument at position 1 must be a non-empty string"
110         checkInvokeResponseFail(t, stub, args, expectedMessage)
111
```

```
112         // It should fail to createDataEntryAd that have one empty arg
113         args = [][]byte{[]byte("createDataEntryAd"),
114                 []byte("2"), []byte(""), []byte("???"), []byte("Unit"),
115                 []byte("20181212152030"), []byte("pub_name"), []byte("10"), []byte("2")}
116         // it should not save to the state and it should fail
117         expectedMessage = "Argument at position 2 must be a non-empty string"
118         checkInvokeResponseFail(t, stub, args, expectedMessage)
119
120         // It should fail to createDataEntryAd that have one empty arg
121         args = [][]byte{[]byte("createDataEntryAd"),
122                 []byte("2"), []byte("test_data"), []byte(""), []byte("Unit"),
123                 []byte("20181212152030"), []byte("pub_name"), []byte("10"), []byte("2")}
124         // it should not save to the state and it should fail
125         expectedMessage = "Argument at position 3 must be a non-empty string"
126         checkInvokeResponseFail(t, stub, args, expectedMessage)
127
128         // It should fail to createDataEntryAd that have one empty arg
129         args = [][]byte{[]byte("createDataEntryAd"),
130                 []byte("2"), []byte("test_data"), []byte("???"), []byte(""),
131                 []byte("20181212152030"), []byte("pub_name"), []byte("10"), []byte("2")}
132         // it should not save to the state and it should fail
133         expectedMessage = "Argument at position 4 must be a non-empty string"
134         checkInvokeResponseFail(t, stub, args, expectedMessage)
135
136         // It should fail to createDataEntryAd that have one empty arg
137         args = [][]byte{[]byte("createDataEntryAd"),
138                 []byte("2"), []byte("test_data"), []byte("???"), []byte("Unit"),
139                 []byte(""), []byte("pub_name"), []byte("10"), []byte("2")}
140         // it should not save to the state and it should fail
141         expectedMessage = "Argument at position 5 must be a non-empty string"
142         checkInvokeResponseFail(t, stub, args, expectedMessage)
```

```
143
144         // It should fail to createDataEntryAd that have one empty arg
145         args = [][]byte{[]byte("createDataEntryAd"),
146                 []byte("2"), []byte("test_data"), []byte("???"), []byte("Unit"),
147                 []byte("20181212152030"), []byte(""), []byte("10"), []byte("2")}
148         // it should not save to the state and it should fail
149         expectedMessage = "Argument at position 6 must be a non-empty string"
150         checkInvokeResponseFail(t, stub, args, expectedMessage)
151
152         // It should fail to createDataEntryAd that have one empty arg
153         args = [][]byte{[]byte("createDataEntryAd"),
154                 []byte("2"), []byte("test_data"), []byte("???"), []byte("Unit"),
155                 []byte("20181212152030"), []byte("pub_name"), []byte(""), []byte("2")}
156         // it should not save to the state and it should fail
157         expectedMessage = "Argument at position 7 must be a non-empty string"
158         checkInvokeResponseFail(t, stub, args, expectedMessage)
159
160         // It should fail to createDataEntryAd that have one empty arg
161         args = [][]byte{[]byte("createDataEntryAd"),
162                 []byte("2"), []byte("test_data"), []byte("???"), []byte("Unit"),
163                 []byte("20181212152030"), []byte("pub_name"), []byte("10"), []byte("")}
164         // it should not save to the state and it should fail
165         expectedMessage = "Argument at position 8 must be a non-empty string"
166         checkInvokeResponseFail(t, stub, args, expectedMessage)
167
168         // It should fail to createDataEntryAd that have less than 8 args
169         args = [][]byte{[]byte("createDataEntryAd"),
170                 []byte("2"), []byte("test_data"), []byte("???"), []byte("Unit"),
171                 []byte("20181212152030"), []byte("pub_name"), []byte("10")}
172         // it should not save to the state and it should fail
173         expectedMessage = "Incorrect number of arguments. Expecting 8"
```

```
174          checkInvokeResponseFail(t, stub, args, expectedMessage)
175
176          // It should fail to createDataEntryAd that have more than 8 args
177          args = [][]byte{[]byte("createDataEntryAd"),
178                  []byte("2"), []byte("test_data"), []byte("???"), []byte("Unit"),
179                  []byte("20181212152030"), []byte("pub_name"), []byte("10"), []byte("2"),
[]byte("2")}
180          // it should not save to the state and it should fail
181          expectedMessage = "Incorrect number of arguments. Expecting 8"
182          checkInvokeResponseFail(t, stub, args, expectedMessage)
183
184          // It should fail to createDataEntryAd for negative price
185          args = [][]byte{[]byte("createDataEntryAd"),
186                  []byte("2"), []byte("test_data"), []byte("???"), []byte("Unit"),
187                  []byte("20181212152030"), []byte("pub_name"), []byte("-10"), []byte("2")}
188          // it should not save to the state and it should fail
189          expectedMessage = "Price cannot be negative number."
190          checkInvokeResponseFail(t, stub, args, expectedMessage)
191
192          // It should fail to createDataEntryAd for negative creationTime
193          args = [][]byte{[]byte("createDataEntryAd"),
194                  []byte("2"), []byte("test_data"), []byte("???"), []byte("Unit"),
195                  []byte("-20181212152030"), []byte("pub_name"), []byte("-10"), []byte("2")}
196          // it should not save to the state and it should fail
197          expectedMessage = "Expecting positiv integer or zero as creation time."
198          checkInvokeResponseFail(t, stub, args, expectedMessage)
199
200          // It should fail to createDataEntryAd if creationTime is not uint
201          args = [][]byte{[]byte("createDataEntryAd"),
202                  []byte("2"), []byte("test_data"), []byte("???"), []byte("Unit"),
203                  []byte("lol"), []byte("pub_name"), []byte("-10"), []byte("2")}
```

```
204         // it should not save to the state and it should fail
205         expectedMessage = "Expecting positiv integer or zero as creation time."
206         checkInvokeResponseFail(t, stub, args, expectedMessage)
207
208         // It should fail to createDataEntryAd if price is not int
209         args = [][]byte{[]byte("createDataEntryAd"),
210                 []byte("2"), []byte("test_data"), []byte("???"), []byte("Unit"),
211                 []byte("20181212152030"), []byte("pub_name"), []byte("lol"), []byte("2")}
212         // it should not save to the state and it should fail
213         expectedMessage = "Expecting positiv integer or zero as price."
214         checkInvokeResponseFail(t, stub, args, expectedMessage)
215 }
216
217 func Test_getDataAdByIDAndTime(t *testing.T) {
218         cc := new(Chaincode)
219         stub := shim.NewMockStub("init_test", cc)
220
221         // Init
222         checkInit(t, stub, [][]byte{[]byte("1")})
223         // create test data entry ad
224         args := [][]byte{[]byte("createDataEntryAd"),
225                 []byte("1"), []byte("test_data"), []byte("10"), []byte("Unit"),
226                 []byte("20181212152030"), []byte("pub_name"), []byte("0"), []byte("2")}
227         // it should save to the state
228         checkInvokeResponse(t, stub, args, "")
229
230         expectedPayload := "{\"RecordType\":\"DATA_ENTRY_AD\",\"DataEntryID\":\"1\"" +
231                 ",\"Description\":\"test_data\",\"Value\":\"10\",\"Unit\":\"Unit\"," +
232                 "\"CreationTime\":20181212152030,\"Publisher\":\"pub_name\"," +
233                 "\"Price\":0,\"AccountNo\":\"2\"}"
234
```

```
   235          // It should get the same expected payload
   236          args = [][]byte{[]byte("getDataAdByIDAndTime"), []byte("1"), []byte("20181212152030")}
   237          checkInvokeResponse(t, stub, args, expectedPayload)
   238
   239          /*
   240                  // This cannot be tested because of the MockStub implementation limitations
   241                  // It should fail to get ID that is not in the ledger
   242                  args = [][]byte{[]byte("getDataAdByID"),
   243                          []byte("2")}
   244                  checkInvokeFail(t, stub, args)
   245          */
   246
   247          // It should fail to getDataAdByIDAndTime if arg is empty string
   248          args = [][]byte{[]byte("getDataAdByIDAndTime"), []byte(""), []byte("20181212152030")}
   249          expectedPayload = "Argument at position 1 must be a non-empty string"
   250          checkInvokeResponseFail(t, stub, args, expectedPayload)
   251
   252          // It should fail to getDataAdByIDAndTime if arg is empty string
   253          args = [][]byte{[]byte("getDataAdByIDAndTime"), []byte("1"), []byte("")}
   254          expectedPayload = "Argument at position 2 must be a non-empty string"
   255          checkInvokeResponseFail(t, stub, args, expectedPayload)
   256
   257          // It should fail to getDataAdByIDAndTime if less than 2 args provided
   258          args = [][]byte{[]byte("getDataAdByIDAndTime"), []byte("1")}
   259          expectedPayload = "Incorrect number of arguments. Expecting data entry Id and
creationTime"
   260          checkInvokeResponseFail(t, stub, args, expectedPayload)
   261
   262          // It should fail to getDataAdByIDAndTime if more than 2 args provided
   263          args = [][]byte{[]byte("getDataAdByIDAndTime"), []byte("1"), []byte("20181212152030"),
[]byte("2")}
```

```
264         expectedPayload = "Incorrect number of arguments. Expecting data entry Id and
creationTime"
265         checkInvokeResponseFail(t, stub, args, expectedPayload)
266
267         // It should fail to getDataAdByIDAndTime if creationTime is not uint
268         args = [][]byte{[]byte("getDataAdByIDAndTime"), []byte("1"), []byte("-20181212152030")}
269         expectedPayload = "Expecting positiv integer or zero as creation time."
270         checkInvokeResponseFail(t, stub, args, expectedPayload)
271
272         // It should fail to getDataAdByIDAndTime if creationTime is not uint
273         args = [][]byte{[]byte("getDataAdByIDAndTime"), []byte("1"), []byte("lol")}
274         expectedPayload = "Expecting positiv integer or zero as creation time."
275         checkInvokeResponseFail(t, stub, args, expectedPayload)
276
277 }
278
279 func Test_getDataAdByPub(t *testing.T) {
280         cc := new(Chaincode)
281         stub := shim.NewMockStub("init_test", cc)
282
283         // Init
284         checkInit(t, stub, [][]byte{[]byte("1")})
285         // create test data entry ad
286         args := [][]byte{[]byte("createDataEntryAd"),
287                 []byte("1"), []byte("test_data"), []byte("???"), []byte("Unit"),
288                 []byte("20181212152030"), []byte("pub_name"), []byte("10"), []byte("2")}
289         // it should save to the state
290         checkInvoke(t, stub, args)
291         expectedPayload := "{\"RecordType\":\"DATA_ENTRY_AD\",\"DataEntryID\":\"1\"" +
292                 ",\"Description\":\"test_data\",\"Value\":\"???\",\"Unit\":\"Unit\"," +
293                 "\"CreationTime\":20181212152030,\"Publisher\":\"pub_name\"," +
```

```
294                 "\"Price\":10,\"AccountNo\":\"2\"}"
295
296         // It should get the same expected payload
297         args = [][]byte{[]byte("getDataAdByPub"), []byte("pub_name")}
298         checkInvokeResponse(t, stub, args, "["+expectedPayload+"]")
299
300         // create second test data entry ad
301         args = [][]byte{[]byte("createDataEntryAd"),
302                 []byte("2"), []byte("test_data"), []byte("100"), []byte("Unit"),
303                 []byte("20181212152030"), []byte("pub_name"), []byte("10"), []byte("2")}
304         // it should save to the state
305         checkInvoke(t, stub, args)
306         expectedPayload2 := "{\"RecordType\":\"DATA_ENTRY_AD\",\"DataEntryID\":\"2\"" +
307                 ",\"Description\":\"test_data\",\"Value\":\"100\",\"Unit\":\"Unit\"," +
308                 "\"CreationTime\":20181212152030,\"Publisher\":\"pub_name\"," +
309                 "\"Price\":10,\"AccountNo\":\"2\"}"
310
311         // It should get both entry as JSON array
312         args = [][]byte{[]byte("getDataAdByPub"), []byte("pub_name")}
313         expectedPayload3 := "[" + expectedPayload + "," + expectedPayload2 + "]"
314         checkInvokeResponse(t, stub, args, expectedPayload3)
315
316         // It should not find entry that is not in state
317         args = [][]byte{[]byte("getDataAdByPub"), []byte("pub_name2")}
318         // expected only empty array
319         expectedPayload = "[]"
320         checkInvokeResponse(t, stub, args, expectedPayload)
321
322         // It should fail with empty arg
323         args = [][]byte{[]byte("getDataAdByPub"), []byte("")}
324         expectedPayload = "Argument at position 1 must be a non-empty string"
```

```
325         checkInvokeResponseFail(t, stub, args, expectedPayload)
326
327         // It should fail with empty more than one arg
328         args = [][]byte{[]byte("getDataAdByPub"), []byte("pub_name"), []byte("pub_name")}
329         expectedPayload = "Incorrect number of arguments. Expecting publisher to get"
330         checkInvokeResponseFail(t, stub, args, expectedPayload)
331 }
332
333 func Test_getAllDataAdByID(t *testing.T) {
334         cc := new(Chaincode)
335         stub := shim.NewMockStub("init_test", cc)
336
337         // Init
338         checkInit(t, stub, [][]byte{[]byte("1")})
339         // create test data entry ad
340         args := [][]byte{[]byte("createDataEntryAd"),
341                 []byte("1"), []byte("test_data"), []byte("???"), []byte("Unit"),
342                 []byte("20181212152030"), []byte("pub_name"), []byte("10"), []byte("2")}
343
344         expectedPayload := "{\"RecordType\":\"DATA_ENTRY_AD\",\"DataEntryID\":\"1\"" +
345                 ",\"Description\":\"test_data\",\"Value\":\"???\",\"Unit\":\"Unit\"," +
346                 "\"CreationTime\":20181212152030,\"Publisher\":\"pub_name\"," +
347                 "\"Price\":10,\"AccountNo\":\"2\"}"
348         // it should save to the state
349         checkInvoke(t, stub, args)
350         args = [][]byte{[]byte("createDataEntryAd"),
351                 []byte("1"), []byte("test_data"), []byte("???"), []byte("Unit"),
352                 []byte("20181212152031"), []byte("pub_name"), []byte("10"), []byte("2")}
353
354         expectedPayload2 := "{\"RecordType\":\"DATA_ENTRY_AD\",\"DataEntryID\":\"1\"" +
355                 ",\"Description\":\"test_data\",\"Value\":\"???\",\"Unit\":\"Unit\"," +
```

```
356                 "\"CreationTime\":20181212152031,\"Publisher\":\"pub_name\"," +
357                 "\"Price\":10,\"AccountNo\":\"2\"}"
358         // it should save to the state
359         checkInvoke(t, stub, args)
360
361         // It should return JSON array with both entries
362         args = [][]byte{[]byte("getAllDataAdByID"), []byte("1")}
363         expectedPayload3 := "[" + expectedPayload + "," + expectedPayload2 + "]"
364         checkInvokeResponse(t, stub, args, expectedPayload3)
365
366         // It should fail with empty arg
367         args = [][]byte{[]byte("getAllDataAdByID"), []byte("")}
368         expectedPayload = "Argument at position 1 must be a non-empty string"
369         checkInvokeResponseFail(t, stub, args, expectedPayload)
370
371         // It should fail with empty more than one arg
372         args = [][]byte{[]byte("getAllDataAdByID"), []byte("1"), []byte("1")}
373         expectedPayload = "Incorrect number of arguments. Expecting data entry Id to get"
374         checkInvokeResponseFail(t, stub, args, expectedPayload)
375 }
376
377 func Test_getLatestDataAdByID(t *testing.T) {
378         cc := new(Chaincode)
379         stub := shim.NewMockStub("init_test", cc)
380
381         // Init
382         checkInit(t, stub, [][]byte{[]byte("1")})
383         // create test data entry ad
384         args := [][]byte{[]byte("createDataEntryAd"),
385                 []byte("1"), []byte("test_data"), []byte("???"), []byte("Unit"),
386                 []byte("20181212152030"), []byte("pub_name"), []byte("10"), []byte("2")}
```

```
387         // it should save to the state
388         checkInvoke(t, stub, args)
389
390         args = [][]byte{[]byte("createDataEntryAd"),
391                 []byte("1"), []byte("test_data"), []byte("???"), []byte("Unit"),
392                 []byte("20181212152031"), []byte("pub_name"), []byte("10"), []byte("2")}
393
394         // it should save to the state
395         checkInvoke(t, stub, args)
396
397         // It should return latest data entry by creationTime
398         args = [][]byte{[]byte("getLatestDataAdByID"), []byte("1")}
399         expectedPayload := "{\"RecordType\":\"DATA_ENTRY_AD\",\"DataEntryID\":\"1\"" +
400                 ",\"Description\":\"test_data\",\"Value\":\"???\",\"Unit\":\"Unit\"," +
401                 "\"CreationTime\":20181212152031,\"Publisher\":\"pub_name\"," +
402                 "\"Price\":10,\"AccountNo\":\"2\"}"
403         checkInvokeResponse(t, stub, args, expectedPayload)
404
405         // It should fail with empty arg
406         args = [][]byte{[]byte("getLatestDataAdByID"), []byte("")}
407         expectedPayload = "Argument at position 1 must be a non-empty string"
408         checkInvokeResponseFail(t, stub, args, expectedPayload)
409
410         // It should fail with empty more than one arg
411         args = [][]byte{[]byte("getLatestDataAdByID"), []byte("1"), []byte("1")}
412         expectedPayload = "Incorrect number of arguments. Expecting data entry Id to get"
413         checkInvokeResponseFail(t, stub, args, expectedPayload)
414 }
415
416 // For this function we cannot test more because of the MockStub limitations
417 func Test_revealPaidData(t *testing.T) {
```

```
418         cc := new(Chaincode)
419         stub := shim.NewMockStub("init_test", cc)
420
421         // Init
422         checkInit(t, stub, [][]byte{[]byte("1")})
423
424         // It should fail to revealPaidData that have less than 7 args
425         args := [][]byte{[]byte("revealPaidData"),
426                 []byte("channel1"), []byte("chaincode_data"), []byte("1"),
[]byte("20181212152030"),
427                 []byte("channel3"), []byte("chaincode_tokens")}
428         expectedMessage := "Incorrect number of arguments. Expecting 7"
429         checkInvokeResponseFail(t, stub, args, expectedMessage)
430
431         // It should fail to revealPaidData that have more than 7 args
432         args = [][]byte{[]byte("revealPaidData"),
433                 []byte("channel1"), []byte("chaincode_data"), []byte("1"),
[]byte("20181212152030"),
434                 []byte("channel3"), []byte("chaincode_tokens"), []byte("TxID-1"),
[]byte("extra_arg")}
435         expectedMessage = "Incorrect number of arguments. Expecting 7"
436         checkInvokeResponseFail(t, stub, args, expectedMessage)
437
438         // It should fail to revealPaidData that have one empty string arg
439         args = [][]byte{[]byte("revealPaidData"),
440                 []byte(""), []byte("chaincode_data"), []byte("1"), []byte("20181212152030"),
441                 []byte("channel3"), []byte("chaincode_tokens"), []byte("TxID-1")}
442         expectedMessage = "Argument at position 1 must be a non-empty string"
443         checkInvokeResponseFail(t, stub, args, expectedMessage)
444
445         // It should fail to revealPaidData that have one empty string arg
```

```
446         args = [][]byte{[]byte("revealPaidData"),
447                 []byte("channel1"), []byte(""), []byte("1"), []byte("20181212152030"),
448                 []byte("channel3"), []byte("chaincode_tokens"), []byte("TxID-1")}
449         expectedMessage = "Argument at position 2 must be a non-empty string"
450         checkInvokeResponseFail(t, stub, args, expectedMessage)
451
452         // It should fail to revealPaidData that have one empty string arg
453         args = [][]byte{[]byte("revealPaidData"),
454                 []byte("channel1"), []byte("chaincode_data"), []byte(""),
[]byte("20181212152030"),
455                 []byte("channel3"), []byte("chaincode_tokens"), []byte("TxID-1")}
456         expectedMessage = "Argument at position 3 must be a non-empty string"
457         checkInvokeResponseFail(t, stub, args, expectedMessage)
458
459         // It should fail to revealPaidData that have one empty string arg
460         args = [][]byte{[]byte("revealPaidData"),
461                 []byte("channel1"), []byte("chaincode_data"), []byte("1"), []byte(""),
462                 []byte("channel3"), []byte("chaincode_tokens"), []byte("TxID-1")}
463         expectedMessage = "Argument at position 4 must be a non-empty string"
464         checkInvokeResponseFail(t, stub, args, expectedMessage)
465
466         // It should fail to revealPaidData that have one empty string arg
467         args = [][]byte{[]byte("revealPaidData"),
468                 []byte("channel1"), []byte("chaincode_data"), []byte("1"),
[]byte("20181212152030"),
469                 []byte(""), []byte("chaincode_tokens"), []byte("TxID-1")}
470         expectedMessage = "Argument at position 5 must be a non-empty string"
471         checkInvokeResponseFail(t, stub, args, expectedMessage)
472
473         // It should fail to revealPaidData that have one empty string arg
474         args = [][]byte{[]byte("revealPaidData"),
```

```
   475                  []byte("channel1"), []byte("chaincode_data"), []byte("1"),
[]byte("20181212152030"),
   476                  []byte("channel3"), []byte(""), []byte("TxID-1")}
   477          expectedMessage = "Argument at position 6 must be a non-empty string"
   478          checkInvokeResponseFail(t, stub, args, expectedMessage)
   479
   480          // It should fail to revealPaidData that have one empty string arg
   481          args = [][]byte{[]byte("revealPaidData"),
   482                  []byte("channel1"), []byte("chaincode_data"), []byte("1"),
[]byte("20181212152030"),
   483                  []byte("channel3"), []byte("chaincode_tokens"), []byte("")}
   484          expectedMessage = "Argument at position 7 must be a non-empty string"
   485          checkInvokeResponseFail(t, stub, args, expectedMessage)
   486
   487          // It should fail to revealPaidData that negative creationTime
   488          args = [][]byte{[]byte("revealPaidData"),
   489                  []byte("channel1"), []byte("chaincode_data"), []byte("1"), []byte("-
20181212152030"),
   490                  []byte("channel3"), []byte("chaincode_tokens"), []byte("TxID-1")}
   491          expectedMessage = "Expecting positiv integer or zero as creation time."
   492          checkInvokeResponseFail(t, stub, args, expectedMessage)
   493
   494          // It should fail to revealPaidData that creationTime is not uint
   495          args = [][]byte{[]byte("revealPaidData"),
   496                  []byte("channel1"), []byte("chaincode_data"), []byte("1"), []byte("lol"),
   497                  []byte("channel3"), []byte("chaincode_tokens"), []byte("TxID-1")}
   498          expectedMessage = "Expecting positiv integer or zero as creation time."
   499          checkInvokeResponseFail(t, stub, args, expectedMessage)
   500
   501  }
   502
```

```
503  // For this function we cannot test more because of the MockStub limitations
504  func Test_checkTXState(t *testing.T) {
505          cc := new(Chaincode)
506          stub := shim.NewMockStub("init_test", cc)
507
508          // Init
509          checkInit(t, stub, [][]byte{[]byte("1")})
510
511          // It should fail to checkTXState that more than 1 arg
512          args := [][]byte{[]byte("checkTXState"), []byte("TxID-1"), []byte("extra_arg")}
513          expectedMessage := "Incorrect number of arguments. Expecting TxID"
514          checkInvokeResponseFail(t, stub, args, expectedMessage)
515
516          // It should fail to checkTXState with empty string arg
517          args = [][]byte{[]byte("checkTXState"), []byte("")}
518          expectedMessage = "Argument at position 1 must be a non-empty string"
519          checkInvokeResponseFail(t, stub, args, expectedMessage)
520  }
521
```

View as plain text