

COBrA DAPP

Source code

Aldo D'Aquino

31 luglio 2018

1 Solidity contracts

BaseContentManagementContract.sol

```
1  pragma solidity ^0.4.0;
2
3  contract CatalogContract {
4      function hasAccess(address u, address x) public view returns(bool);
5      function consumeContent(address u) public;
6      function addMe() public;
7      function removesMe() public;
8  }
9
10 contract BaseContentManagementContract {
11
12     /* VARIABLES */
13
14     // Runtime
15     address public catalog;
16     address public owner;
17     bytes32 public name;
18     bytes32 public genre;
19     uint public price = 0; // is assumed that the content can be free,
20     //and so the price is 0.
21     bool private published = false;
22     CatalogContract private catalogContract;
23
24
25     /* EVENTS */
26     event ContentPublished();
27     event ContentDeleted();
28     event contentConsumed(address user);
29
30
31     /* MODIFIERS */
32     modifier onlyOwner() {
33         require(msg.sender == owner);
34         _;
35     }
36
37
38     /* FUNCTIONS */
39     /** Constructor */
40     constructor() public {
41         owner = msg.sender;
42     }
43
44     /** Fallback function */
45     function () public {
46         revert();
47     }
48
49     /** Suicide function, can be called only by the owner */
50     function _suicide() public onlyOwner {
51         // notice the catalog
52         catalogContract.removesMe();
53         // emit an event
54         emit ContentDeleted();
55         // if there is some wei send it to the author
56         selfdestruct(owner);
```

```

57     }
58
59     /** Suicide function, can be called only by the owner */
60     function murder() public {
61         require(msg.sender == catalog);
62         // emit an event
63         emit ContentDeleted();
64         // if there is some wei send it to the author
65         selfdestruct(owner);
66     }
67
68     /** Used by the customers to consume this content after requesting the
69     * access.
70     * @return the content.
71     */
72     function consumeContent() public returns(bytes) {
73         require(published);
74         require(catalogContract.hasAccess(msg.sender, this));
75         catalogContract.consumeContent(msg.sender);
76         emit contentConsumed(msg.sender);
77     }
78
79     /** Used by the author to publish the content.
80     * @param c the address of the catalog in which publish the content.
81     * The author must specify the name of this content before calling this
82     * function.
83     * Can be called only one time.
84     */
85     function publish(address c) public onlyOwner {
86         require(!published);
87         require(name[0] != 0);
88         published = true;
89         catalog = c;
90         catalogContract = CatalogContract(c);
91         catalogContract.addMe();
92         emit ContentPublished();
93     }
94 }

```

CatalogContract.sol

```

1  pragma solidity ^0.4.0;
2
3  contract BaseContentManagementContract {
4      address public owner;
5      bytes32 public name;
6      bytes32 public genre;
7      uint public price;
8      function murder() public;
9  }
10
11  contract CatalogContract {
12
13      /* VARIABLES */
14      // Constants
15      //uint public contentCost = 0.01 ether;    // ~ 4€ - deprecated: now the price is chosen by
16      ↳ the author
17      uint public premiumCost = 0.1 ether;    // ~ 40€

```

```

17 uint public premiumTime = 172800;           // ~ 1 month
18 uint public payAfter = 2; // views - IMPORTANT NOTE: should be 10, but is set to 2 for
    ↳ testing.
19
20 bytes32 private enjoyS = "enjoy";
21 bytes32 private valueForMoneyS = "value for money";
22 bytes32 private contentS = "content";
23 bytes32[] public ratingCategories = [enjoyS, valueForMoneyS, contentS];
24
25 // Runtime
26 address public owner;
27 uint private balance = 0;
28
29 // Structs
30 struct content {
31     bytes32 name;
32     address author;
33     bytes32 genre;
34     uint price;
35     uint views;
36     uint uncollectedViews;
37     // Feedback category:
38     // - how much do you enjoy the content (personal opinion)
39     // - price quality ratio
40     // - how do you think the content is good (objective opinion, based on
41     // the meaning it would like to have)
42     uint enjoySum;
43     uint enjoyNum;
44     uint valueForMoneySum;
45     uint valueForMoneyNum;
46     uint contentMeaningSum;
47     uint contentMeaningNum;
48 }
49
50 struct author {
51     bool alreadyFound;
52     uint amount;
53     uint ratio;
54 }
55
56 // map a user into his subscription expiration time
57 mapping (address => uint) private premiumUsers;
58 // map a user into his accessible contents
59 mapping (address => mapping (address => bool)) private accessibleContent;
60 address[] contentList; // list of all contents
61 // map content addresses into contents
62 mapping (address => content) private contents;
63 // map a user into the content that he can vote
64 mapping (address => mapping (address => bool)) private pendingFeedback;
65
66 // Support structure for suicide function
67 address[] private authorsList; // list of all authors
68 mapping (address => author) private authors;
69
70
71 /* EVENTS */
72 event CatalogClosed();
73 event GrantedAccess(address user, address content);
74 event PaymentAvailable(address content);

```

```

75 event BecomesPremium(address user);
76 event NewContentAvailable(bytes32 name, address addr);
77 event FeedbackAvailable(address content, address user);
78
79
80 /* MODIFIERS */
81 modifier onlyOwner() {
82     require(msg.sender == owner);
83     -;
84 }
85
86 modifier exists(address c) {
87     require(contents[c].name != "" &&
88         BaseContentManagementContract(c).owner() != 0);
89     -;
90 }
91
92
93 /* FUNCTIONS */
94
95 /** Constructor */
96 constructor() public {
97     owner = msg.sender;
98 }
99
100 /** Fallback function */
101 function () public {
102     revert();
103 }
104
105 /** Suicide function, can be called only by the owner */
106 function _suicide() public onlyOwner {
107     uint totalRatio = 0;
108     for (uint i = 0; i < contentList.length; i++) {
109         content memory cc = contents[contentList[i]];
110         // save the author in the list if not exist
111         if (!authors[cc.author].alreadyFound) {
112             authors[cc.author].alreadyFound = true;
113             authorsList.push(cc.author);
114         }
115         // calculate the payout rate for this author
116         uint enjoyRate = 0;
117         uint priceFairnessRate = 0;
118         uint contentMeaningRate = 0;
119         if (cc.enjoyNum != 0) enjoyRate = cc.enjoySum / cc.enjoyNum;
120         if (cc.valueForMoneyNum != 0)
121             priceFairnessRate = cc.valueForMoneySum / cc.valueForMoneyNum;
122         if (cc.contentMeaningNum != 0) contentMeaningRate =
123             cc.contentMeaningSum / cc.contentMeaningNum;
124         uint average_rate = (enjoyRate + priceFairnessRate +
125             contentMeaningRate) / 3;
126         if (average_rate == 0)
127             average_rate = 5; // if no one have voted just cast to max
128         // add the payout for the uncollected views to yhe author amount
129         uint amount = cc.uncollectedViews * cc.price * average_rate / 5;
130         authors[cc.author].amount += amount;
131         // remove the amount from the balance (later we will pay this
132         // amount, so we cannot count it in the balance)
133         balance -= amount;

```

```

134     // We calculate the rating in percentage (uint => no decimals => we
135     // need precision). Then we multiply this rate for the price of the
136     // content and the number of views. Then we sum it all together.
137     // What we obtain is a number that grows linearly basing on the
138     // number of content of this author, the price of this contents, the
139     // view count and the rating, so that who do more and better
140     // receives more.
141     uint ratio = cc.views * cc.price * 100 * average_rate * 25;
142     // 25 = 100 / 5 where 100 is the percentage and 5 is the maximum rate.
143     authors[cc.author].ratio += ratio;
144     // We keep track also of the total ratio off all the authors.
145     totalRatio += ratio;
146     // Reset the number of views and uncollected views of this content.
147     // We don't need it anymore. This help to prevent reentrancy.
148     contents[contentList[i]].views = 0;
149     contents[contentList[i]].uncollectedViews = 0;
150     // Murder all the contents in the catalog: this will free up space
151     // in the blockchain and create negative gas to consume less in this
152     // process: all this transfers cost a lot.
153     BaseContentManagementContract(contentList[i]).murder();
154 }
155 if (totalRatio != 0) { // avoid division by 0
156     // Distribute the balance to the authors according with their ratio.
157     // We calculate the ratio between totalRatio and a.ratio, then
158     // multiply this value for the balance. This gave us how many part
159     // of the balance belongs to the author. We add it to the author
160     // amount.
161     for (i = 0; i < authorsList.length; i++) {
162         author memory a = authors[authorsList[i]];
163         amount = a.ratio * balance / totalRatio;
164         uint toBeTransferred = a.amount = amount;
165         balance -= amount;
166         // reset the author to prevent reentrancy
167         authors[authorsList[i]].amount = 0;
168         authors[authorsList[i]].ratio = 0;
169         // transfer the amount to the owner
170         if (toBeTransferred != 0) authorsList[i].transfer(toBeTransferred);
171     }
172 }
173 // emit an event
174 emit CatalogClosed();
175 // Transfer weis in excess to the owner
176 selfdestruct(owner);
177 }
178
179 /** Pays for access to content x.
180  * @param x the address of the block of the ContentManagementContract.
181  * Gas: who requests the content pays.
182  */
183 function getContent(address x) public payable exists(x) {
184     grantAccess(msg.sender, x);
185 }
186
187 /** Pays for granting access to content x to the user u.
188  * @param x the address of the block of the ContentManagementContract.
189  * @param u the user to whom you want to gift the content.
190  * Gas: who gift pays.
191  */
192 function giftContent(address x, address u) public payable exists(x) {

```

```

193     grantAccess(u, x);
194 }
195
196 /** Pays for granting a Premium Account to the user u.
197 * @param u the user to whom you want to gift the subscription.
198 * Gas: who gift pays.
199 */
200 function giftPremium(address u) public payable {
201     setPremium(u);
202 }
203
204 /** Starts a new premium subscription.
205 * Gas: who subscribe pays.
206 */
207 function buyPremium() public payable {
208     setPremium(msg.sender);
209 }
210
211 /** Leave a feedback on a content.
212 * @param c the content address.
213 * @param y the category in which leave the feedback.
214 * Can be "enjoy", "value for money" or "content".
215 * @param r the vote that you want to assign, from 1 to 5.
216 */
217 function leaveFeedback(address c, bytes32 y, uint r) public {
218     require(r <= 5 && pendingFeedback[msg.sender][c]);
219     if (y == "enjoy") {
220         contents[c].enjoySum += r;
221         contents[c].enjoyNum++;
222     }
223     if (y == "value for money") {
224         contents[c].valueForMoneySum += r;
225         contents[c].valueForMoneyNum++;
226     }
227     if (y == "content") {
228         contents[c].contentMeaningSum += r;
229         contents[c].contentMeaningNum++;
230     }
231     pendingFeedback[msg.sender][c] = false;
232 }
233
234 /** Used to know the reached payout of a content.
235 * @param x the content.
236 * @return the reached payout
237 * or 0 if the content does not have received enough views.
238 * Gas: the author (who receives money) pays.
239 */
240 function payoutAvailable(address x) public view
241 returns(uint) {
242     content memory c = contents[x];
243     uint uncollectedViews = c.uncollectedViews;
244     if (uncollectedViews < payAfter) return 0;
245     uint enjoyRate = 0;
246     uint priceFairnessRate = 0;
247     uint contentMeaningRate = 0;
248     if (c.enjoyNum != 0) enjoyRate = c.enjoySum / c.enjoyNum;
249     if (c.valueForMoneyNum != 0)
250         priceFairnessRate = c.valueForMoneySum / c.valueForMoneyNum;
251     if (c.contentMeaningNum != 0)

```

```

252         contentMeaningRate = c.contentMeaningSum / c.contentMeaningNum;
253         uint average_rate = (enjoyRate + priceFairnessRate +
254         contentMeaningRate) / 3;
255         if (average_rate == 0)
256             average_rate = 5;    // if no one have voted just cast to max
257         uint amount = c.price * uncollectedViews * average_rate / 5;
258         return amount;
259     }
260
261     /** Used by the authors to collect their reached payout.
262     * The content must has been visited at least payAfter times.
263     * @param x the content.
264     * (the author should have received the event).
265     * Gas: the author (who receives money) pays.
266     */
267     function collectPayout(address x) public {
268         require (contents[x].author == msg.sender);
269         uint amount = payoutAvailable(x);
270         require(amount > 0);
271         contents[x].uncollectedViews = 0;
272         balance -= amount;
273         msg.sender.transfer(amount);
274     }
275
276     /** Called from a ContentManagementContract.
277     * Adds the content to the catalog.
278     * Gas: the author pays.
279     */
280     function addMe() public {
281         BaseContentManagementContract cc =
282         BaseContentManagementContract(msg.sender);
283         contents[cc] = content(cc.name(), cc.owner(), cc.genre(), cc.price(),
284         0, 0, 0, 0, 0, 0, 0, 0, 0);
285         contentList.push(cc);
286         emit NewContentAvailable(cc.name(), cc);
287     }
288
289     /** Notice the catalog that the user u has consumed the content x.
290     * @param u the user that consume the content.
291     * Gas: the user that consumes the content pays.
292     */
293     function consumeContent(address u) public exists(msg.sender) {
294         // Premium users can consume contents for free and are not considered
295         // in the count of views and rating. This is required by the
296         // assignment, but also ensure that authors cannot downvote other
297         // authors contents without paying.
298         if (isPremium(u)) return;
299         // Only contents can call this function, so the content to be delete
300         // is the msg.sender
301         delete accessibleContent[u][msg.sender];
302         pendingFeedback[u][msg.sender] = true;
303         emit FeedbackAvailable(msg.sender, u);
304         contents[msg.sender].views++;
305         contents[msg.sender].uncollectedViews++;
306         /* Notice the author if his contents has enough views.
307         * Note that the event is emitted only once, when the number of views
308         * is exactly equal to payAfter: it is not an oversight but a caution
309         * not to spam too much. Can be changed in >= if this contract is
310         * deployed in a dedicated blockchain. */

```



```

311         if (contents[msg.sender].uncollectedViews == payAfter) {
312             emit PaymentAvailable(msg.sender);
313         }
314     }
315
316     /** Called from a ContentManagementContract, removes the content from the
317     * catalog (used by the suicide function).
318     * Gas: the author pays.
319     */
320     function removesMe() public exists(msg.sender) {
321         delete contents[msg.sender];
322         bool found = false;
323         // Search the address in the array
324         for (uint i = 0; i < contentList.length; i++) {
325             // lazy if: skip the storage read if found is true
326             if (!found && contentList[i] == msg.sender) {
327                 found = true;
328             }
329             if (found && i < contentList.length - 1) {
330                 // move all the following items back of 1 position
331                 contentList[i] = contentList[i+1];
332             }
333         }
334         if (found) {
335             // and finally delete the last item
336             delete contentList[contentList.length - 1];
337             contentList.length--;
338         }
339     }
340
341     /** Returns the number of views for each content.
342     * @return (bytes32[], uint[], address[]), names, addresses and views:
343     * each content in names is associated with the views number in views and
344     * with its address in addresses.
345     * Gas: no one pay.
346     * Burden: O(n).
347     */
348     function getStatistics() public view returns(bytes32[], address[], uint[]) {
349         bytes32[] memory names = new bytes32[](contentList.length);
350         uint[] memory views = new uint[](contentList.length);
351         for (uint i = 0; i < contentList.length; i++) {
352             content memory c = contents[contentList[i]];
353             names[i] = c.name;
354             views[i] = c.views;
355         }
356         return (names, contentList, views);
357     }
358
359     /** Returns the list of contents without the number of views.
360     * @return (string[], address[]) names and addresses: each content in names
361     * is associated with its address in addresses.
362     * Gas: no one pay.
363     * Burden: O(n).
364     */
365     function getContentList() public view returns(bytes32[], address[]) {
366         bytes32[] memory names = new bytes32[](contentList.length);
367         for (uint i = 0; i < contentList.length; i++) {
368             names[i] = contents[contentList[i]].name;
369         }

```

```

370         return (names, contentList);
371     }
372
373     /** Returns the list of contents with all information.
374     * @return (address[], bytes32[], address[], bytes32[], uint[], uint[]).
375     * In the position n we got in order address,
376     * name, author, genre, price, views of the content n.
377     * Gas: no one pay.
378     * Burden: O(n).
379     */
380     function getFullContentList() public view
381     returns(address[], bytes32[], address[], bytes32[], uint[], uint[]) {
382         bytes32[] memory name = new bytes32[](contentList.length);
383         address[] memory authorAddr = new address[](contentList.length);
384         bytes32[] memory genre = new bytes32[](contentList.length);
385         uint[] memory price = new uint[](contentList.length);
386         uint[] memory views = new uint[](contentList.length);
387         for (uint i = 0; i < contentList.length; i++) {
388             content memory c = contents[contentList[i]];
389             name[i] = c.name;
390             authorAddr[i] = c.author;
391             genre[i] = c.genre;
392             price[i] = c.price;
393             views[i] = c.views;
394         }
395         return (contentList, name, authorAddr, genre, price, views);
396     }
397
398     /** Returns ratings list of contents.
399     * @return (address[], uint[], uint[], uint[], uint[]). In the position n we
400     * got in order address, average rating enjoy rating, value for money rating
401     * and content meaning rating of the content n.
402     * Gas: no one pay.
403     * Burden: O(n).
404     */
405     function getRatingsList() public view
406     returns(address[], uint[], uint[], uint[], uint[]) {
407         uint[] memory averageRating = new uint[](contentList.length);
408         uint[] memory enjoy = new uint[](contentList.length);
409         uint[] memory priceFairness = new uint[](contentList.length);
410         uint[] memory contentMeaning = new uint[](contentList.length);
411         for (uint i = 0; i < contentList.length; i++) {
412             content memory c = contents[contentList[i]];
413             if (c.enjoyNum != 0)
414                 enjoy[i] = c.enjoySum / c.enjoyNum;
415             if (c.valueForMoneyNum != 0)
416                 priceFairness[i] = c.valueForMoneySum / c.valueForMoneyNum;
417             if (c.contentMeaningNum != 0)
418                 contentMeaning[i] = c.contentMeaningSum / c.contentMeaningNum;
419             averageRating[i] =
420                 (enjoy[i] + priceFairness[i] + contentMeaning[i]) / 3;
421         }
422         return (contentList, averageRating, enjoy, priceFairness,
423             contentMeaning);
424     }
425
426     /** Returns all the information about a content.
427     * @param addr address of the content.
428     * @return (bytes32, address, bytes32, uint, uint) corresponding to name,

```

```

429     * author, genre, price and views of the content.
430     * Gas: no one pay.
431     * Burden:  $O(n)$ .
432     */
433 function getContentInfo(address addr) public view
434 returns(bytes32, address, bytes32, uint, uint) {
435     content memory c = contents[addr];
436     return (c.name, c.author, c.genre, c.price, c.views);
437 }
438
439 /** Returns ratings for a content.
440     * @param addr address of the content.
441     * @return returns(uint, uint, uint, uint) corresponding to total, enjoy,
442     * priceFairness and contentMeaning rating of the content.
443     * Gas: no one pay.
444     * Burden:  $O(n)$ .
445     */
446 function getContentRatings(address addr) public view
447 returns(uint, uint, uint, uint) {
448     content memory c = contents[addr];
449     uint total = 0;
450     uint enjoy = 0;
451     uint priceFairness = 0;
452     uint contentMeaning = 0;
453     if (c.enjoyNum != 0)
454         enjoy = c.enjoySum / c.enjoyNum;
455     if (c.valueForMoneyNum != 0)
456         priceFairness = c.valueForMoneySum / c.valueForMoneyNum;
457     if (c.contentMeaningNum != 0)
458         contentMeaning = c.contentMeaningSum / c.contentMeaningNum;
459     total = (enjoy + priceFairness + contentMeaning) / 3;
460     return (total, enjoy, priceFairness, contentMeaning);
461 }
462
463 /** Returns the list of n newest contents.
464     * @param n the number of item that you want in the list.
465     * @return (string[], address[]) names and addresses ordered from the
466     * newest: each content in names is associated with its address in
467     * addresses.
468     * Gas: no one pay.
469     * Burden:  $O(x) \sim O(1)$ .
470     */
471 function getNewContentList(uint n) public view
472 returns(bytes32[], address[]) {
473     uint listLength = n;
474     // If i have less than chartListLength element in the contentList I
475     // have to return contentList.length elements
476     if (contentList.length < listLength) listLength = contentList.length;
477     // NOTE: I assume that the latest content is not the last deployed
478     // contract in the blockchain (with the highest block number), but is
479     // the last added to the catalog (that ideally is when is "published").
480     bytes32[] memory names = new bytes32[](listLength);
481     address[] memory addresses = new address[](listLength);
482     for (uint i = 0; i < listLength; i++) {
483         // add it in reverse order: the latest first
484         address a = contentList[contentList.length - 1 - i];
485         names[i] = contents[a].name;
486         addresses[i] = a;
487     }

```

```

488     return (names, addresses);
489 }
490
491 /** Get the latest release of genre g.
492  * @param g the genre of which you want to get the latest content.
493  * @return (bytes32, address) names and addresses of the content.
494  * Gas: no one pay.
495  * Burden: < O(n).
496  */
497 function getLatestByGenre(bytes32 g) public view returns(bytes32, address) {
498     // using int because i can be negative if the list is empty or there
499 // aren't element of genre g. Should not fail.
500     int i = int(contentList.length - 1);
501     while (i >= 0) {
502         address addr = contentList[uint(i)];
503         content memory c = contents[addr];
504         if (c.genre == g) {
505             return (c.name, addr);
506         }
507         i--;
508     }
509     // fallback, return empty if not exist a release of g
510     return("", 0);
511 }
512
513 /** Get the latest release of the author a.
514  * @param a the author of whom you want to get the latest content.
515  * @return (bytes32, address) names and addresses of the content.
516  * Gas: no one pay.
517  * Burden: < O(n).
518  */
519 function getLatestByAuthor(address a) public view
520 returns(bytes32, address) {
521     // using int because i can be negative if the list is empty or there
522 // aren't element of genre g. Should not fail.
523     int i = int(contentList.length - 1);
524     while (i >= 0) {
525         address addr = contentList[uint(i)];
526         content memory c = contents[addr];
527         if (c.author == a) {
528             return (c.name, addr);
529         }
530         i--;
531     }
532     // fallback, return empty if not exist a release of a
533     return("", 0);
534 }
535
536 /** Get the most popular content.
537  * @return (string, address) name and address of the content.
538  * If there are 2 or more content with the same number of view the oldest
539  * comes first.
540  * Gas: no one pay.
541  * Burden: O(n).
542  */
543 function getMostPopular() public view
544 returns(bytes32, address) {
545     int maxViews = -1;
546     bytes32 maxName;

```

```

547     address maxAddress;
548     for (uint i = 0; i < contentList.length; i++) {
549         address addr = contentList[i];
550         content memory c = contents[addr];
551         if (int(c.views) > maxViews) {
552             maxViews = int(c.views);
553             maxName = c.name;
554             maxAddress = addr;
555         }
556     }
557     return (maxName, maxAddress);
558 }
559
560 /** Get the most popular release of genre g.
561  * @param g the genre of which you want to get the most popular content.
562  * @return (string, address) name and address of the content.
563  * If there are 2 or more content with the same number of view the oldest
564  * comes first.
565  * Gas: no one pay.
566  * Burden: O(n).
567  */
568 function getMostPopularByGenre(bytes32 g) public view
569 returns(bytes32, address) {
570     int maxViews = -1;
571     bytes32 maxName;
572     address maxAddress;
573     for (uint i = 0; i < contentList.length; i++) {
574         address addr = contentList[i];
575         content memory c = contents[addr];
576         if (c.genre == g && int(c.views) > maxViews) {
577             maxViews = int(c.views);
578             maxName = c.name;
579             maxAddress = addr;
580         }
581     }
582     return (maxName, maxAddress);
583 }
584
585 /** Get the most popular release of the author a.
586  * @param a the author of which you want to get the most popular content.
587  * @return (string, address) name and address of the content.
588  * If there are 2 or more content with the same number of view the oldest
589  * comes first.
590  * Gas: no one pay.
591  * Burden: O(n).
592  */
593 function getMostPopularByAuthor(address a) public view
594 returns(bytes32, address) {
595     int maxViews = -1;
596     bytes32 maxName;
597     address maxAddress;
598     for (uint i = 0; i < contentList.length; i++) {
599         address addr = contentList[i];
600         content memory c = contents[addr];
601         if (c.author == a && int(c.views) > maxViews) {
602             maxViews = int(c.views);
603             maxName = c.name;
604             maxAddress = addr;
605         }

```

```

606     }
607     return (maxName, maxAddress);
608 }
609
610 /** Get the release with highest rating in category y.
611 * @param y the category, optional. If not specified returns the content
612 * with the maximum average rating.
613 * @return (string, address) name and address of the content.
614 * If there are 2 or more content with the same number of view the oldest
615 * comes first.
616 * Gas: no one pay.
617 * Burden: O(n).
618 */
619 function getMostRated(bytes32 y) public view returns(bytes32, address) {
620     int maxRate = -1;
621     bytes32 maxName;
622     address maxAddress;
623     for (uint i = 0; i < contentList.length; i++) {
624         address addr = contentList[i];
625         content memory c = contents[addr];
626         uint rate = 0;
627         if ((y[0] == 0 || y == "enjoy") && c.enjoyNum != 0) {
628             rate += c.enjoySum / c.enjoyNum;
629         }
630         if ((y[0] == 0 || y == "value for money")
631             && c.valueForMoneyNum != 0) {
632             rate += c.valueForMoneySum / c.valueForMoneyNum;
633         }
634         if ((y[0] == 0 || y == "content") && c.contentMeaningNum != 0) {
635             rate += c.contentMeaningSum / c.contentMeaningNum;
636         }
637         if (int(rate) > maxRate) {
638             maxRate = int(rate);
639             maxName = c.name;
640             maxAddress = addr;
641         }
642     }
643     return (maxName, maxAddress);
644 }
645
646 /** Get the release with highest rating in category y with genre g.
647 * @param g the genre.
648 * @param y the category, optional. If not specified returns the content
649 * with the maximum average rating.
650 * @return (string, address) name and address of the content.
651 * If there are 2 or more content with the same number of view the oldest
652 * comes first.
653 * Gas: no one pay.
654 * Burden: O(n).
655 */
656 function getMostRatedByGenre(bytes32 g, bytes32 y) public view
657 returns(bytes32, address) {
658     int maxRate = -1;
659     bytes32 maxName;
660     address maxAddress;
661     for (uint i = 0; i < contentList.length; i++) {
662         address addr = contentList[i];
663         content memory c = contents[addr];
664         if (c.genre == g) {

```

```

665         uint rate = 0;
666         if ((y[0] == 0 || y == "enjoy") && c.enjoyNum != 0) {
667             rate += c.enjoySum / c.enjoyNum;
668         }
669         if ((y[0] == 0 || y == "value for money")
670             && c.valueForMoneyNum != 0) {
671             rate += c.valueForMoneySum / c.valueForMoneyNum;
672         }
673         if ((y[0] == 0 || y == "content") && c.contentMeaningNum != 0) {
674             rate += c.contentMeaningSum / c.contentMeaningNum;
675         }
676         if (int(rate) > maxRate) {
677             maxRate = int(rate);
678             maxName = c.name;
679             maxAddress = addr;
680         }
681     }
682 }
683 }
684 return (maxName, maxAddress);
685 }
686
687 /** Get the release with highest rating in category y by author a.
688 * @param a the author.
689 * @param y the category, optional. If not specified returns the content
690 * with the maximum average rating.
691 * @return (string, address) name and address of the content.
692 * If there are 2 or more content with the same number of view the oldest
693 * comes first.
694 * Gas: no one pay.
695 * Burden: O(n).
696 */
697 function getMostRatedByAuthor(address a, bytes32 y) public view
698 returns(bytes32, address) {
699     int maxRate = -1;
700     bytes32 maxName;
701     address maxAddress;
702     for (uint i = 0; i < contentList.length; i++) {
703         address addr = contentList[i];
704         content memory c = contents[addr];
705         if (c.author == a) {
706             uint rate = 0;
707             if ((y[0] == 0 || y == "enjoy") && c.enjoyNum != 0) {
708                 rate += c.enjoySum / c.enjoyNum;
709             }
710             if ((y[0] == 0 || y == "value for money")
711                 && c.valueForMoneyNum != 0) {
712                 rate += c.valueForMoneySum / c.valueForMoneyNum;
713             }
714             if ((y[0] == 0 || y == "content") && c.contentMeaningNum != 0) {
715                 rate += c.contentMeaningSum / c.contentMeaningNum;
716             }
717             if (int(rate) > maxRate) {
718                 maxRate = int(rate);
719                 maxName = c.name;
720                 maxAddress = addr;
721             }
722         }
723     }

```

```

724     }
725     return (maxName, maxAddress);
726 }
727
728 /** Checks if a user u has access to a content x.
729     * @param u the user of whom you want to check the access right.
730     * @param x the content of which you want to check the access right.
731     * @return bool true if the user has the access right, false otherwise.
732     * Gas: no one pay.
733     * Burden: small.
734     */
735 function hasAccess(address u, address x) public view exists(x)
736 returns(bool) {
737     // lazy or, premium first because we suppose they consume more content
738     // than standard users
739     return isPremium(u) || accessibleContent[u][x];
740 }
741
742 /** Checks if a user u has an active premium subscription.
743     * @param u the user of whom you want to check the premium subscription.
744     * @return bool true if the user hold a still valid premium account, false
745     * otherwise.
746     * Gas: no one pay.
747     * Burden: small.
748     */
749 function isPremium(address u) public view returns(bool) {
750     return premiumUsers[u] >= block.number;
751 }
752
753
754 /* INTERNAL AUXILIARY FUNCTIONS */
755
756 /** Starts a new premium subscription for the user u based on the amount v.
757     * @param u the user.
758     */
759 function setPremium(address u) private {
760     require(msg.value == premiumCost);
761     // If the user has never bought premium or the premium subscription is
762     // expired reset the expiration time to now
763     if (!isPremium(u)) premiumUsers[u] = block.number;
764     // Increment the user expiration time
765     // (if he is already premium will be premium longer)
766     premiumUsers[u] += premiumTime;
767     emit BecomesPremium(u);
768     balance += msg.value;
769 }
770
771 /** Grant access for the content x to the user v.
772     * @param u the user.
773     * @param x the content.
774     */
775 function grantAccess(address u, address x) private {
776     // do not manage the extra value,
777     // just require exactly what the content cost
778     require(msg.value == contents[x].price);
779     // prevent double purchase of contents
780     require(!accessibleContent[u][x]);
781     // the author cannot buy his contents
782     // this also ensure that an author cannot vote its content to increase

```



```

783         // the withdrawal
784         require(contents[x].author != u);
785         // grant access
786         accessibleContent[u][x] = true;
787         emit GrantedAccess(u, x);
788         // update balance
789         balance += msg.value;
790     }
791 }

```

DAPPContentManagementContract.sol

```

1  pragma solidity ^0.4.0;
2
3  import "./GenericContentManagementContract.sol";
4
5  contract DAPPContentManagementContract is GenericContentManagementContract {
6
7      bytes32 public hostname;
8      uint public port;
9
10     /** Used by the author to set the server hostname.
11      */
12     function setHostname(bytes32 h) public onlyOwner {
13         hostname = h;
14     }
15
16     /** Used by the author to set the server port.
17      */
18     function setPort(uint p) public onlyOwner {
19         port = p;
20     }
21
22     /** Used by the author to publish the content.
23      * @param c the address of the catalog in which publish the content.
24      * The author must specify the name of this content and the author-server
25      * hostname and port before calling this function.
26      * Can be called only one time.
27      */
28     function publish(address c) public onlyOwner {
29         require(hostname[0] != 0 && port != 0);
30         super.publish(c);
31     }
32 }

```

GenericContentManagementContract.sol

```

1  pragma solidity ^0.4.0;
2
3  import "./BaseContentManagementContract.sol";
4
5  contract GenericContentManagementContract is BaseContentManagementContract {
6
7      /** Used by the author to set the name.
8       * Can be called only one time.
9       */
10     function setName(bytes32 n) public onlyOwner {

```

```

11         require(name[0] == 0);
12         name = n;
13     }
14
15     /** Used by the author to set the genre.
16     * Can be called only one time, but its call is not mandatory (the content can not have a
17     genre).
18     */
19     function setGenre(bytes32 g) public onlyOwner {
20         require(genre[0] == 0);
21         genre = g;
22     }
23
24     /** Used by the author to set the content price.
25     * Can be called only one time.
26     */
27     function setPrice(uint p) public onlyOwner {
28         require(price == 0);
29         price = p;
30     }

```

2 DAPP

2.1 DAPP/author-server

Main.java

```
1  package com.aldodaquino.cobra.authorserver;
2
3  import java.io.File;
4  import java.io.IOException;
5  import java.math.BigInteger;
6  import java.net.InetSocketAddress;
7  import java.nio.channels.ServerSocketChannel;
8  import java.util.HashMap;
9  import java.util.Map;
10
11 import com.aldodaquino.cobra.connections.Status;
12 import com.aldodaquino.javaults.FileExchange;
13 import com.aldodaquino.cobra.connections.CobraHttpHelper;
14 import com.aldodaquino.cobra.main.CatalogManager;
15 import com.aldodaquino.cobra.main.ContentManager;
16 import com.aldodaquino.javaults.CliHelper;
17 import com.sun.net.httpserver.HttpExchange;
18 import com.sun.net.httpserver.HttpServer;
19
20 import org.web3j.crypto.Credentials;
21
22 /**
23  * The Main class of the author server. The author server remains always online to listen
24  * ↪ access request. The author can
25  * use its server to deploy his content on the blockchain and publish it on the catalog. This
26  * ↪ server serves content also
27  * when the author has closed his client.
28  * Includes two functions that handle the two urls /deploy (to deploy a content) and /access to
29  * ↪ access a content.
30  * @author Aldo D'Aquino.
31  * @version 1.0.
32  */
33 public class Main {
34
35     private static final int DEFAULT_PORT = 8080;
36     private static final String CONTENT_FILE_PATH = "author_content_files/";
37
38     /**
39      * Main method.
40      * @param args a String[] passed through command line.
41      * @throws IOException if it is not possible to create the server.
42      */
43     public static void main(String[] args) throws IOException {
44
45         // Parse cmd options
46         CliHelper cliHelper = new CliHelper();
47         cliHelper.addOption("h", "help", false, "Print this help message.");
48         cliHelper.addOption("k", "private-key", true,
49             "Private key of your account (required).");
50         cliHelper.addOption("c", "catalog", true, "Catalog address (required).");
51         cliHelper.addOption("n", "hostname", true,
52             "Name of this host, i.e. the IP address of this server, used to deploy content
53             ↪ (required).");
54         cliHelper.addOption("p", "port", true,
```

```

51         "Port on which run the server. Default: " + DEFAULT_PORT + ".");
52 cliHelper.parse(args);
53
54 if (cliHelper.isPresent("h")) {
55     System.out.println(cliHelper.getHelpMessage());
56     System.exit(0);
57 }
58
59 Status status = new Status();
60
61 status.privateKey = cliHelper.getValue("private-key");
62 if (status.privateKey == null || status.privateKey.length() == 0) {
63     System.err.println(cliHelper.getMissingOptionMessage("private-key"));
64     System.err.flush();
65     System.out.println(cliHelper.getHelpMessage());
66     System.out.flush();
67     System.exit(1);
68 }
69
70 String catalogAddress = cliHelper.getValue("catalog");
71 if (catalogAddress == null || catalogAddress.length() == 0) {
72     System.err.println(cliHelper.getMissingOptionMessage("catalog"));
73     System.err.flush();
74     System.out.println(cliHelper.getHelpMessage());
75     System.out.flush();
76     System.exit(1);
77 }
78
79 status.hostname = cliHelper.getValue("hostname");
80 if (status.hostname == null || status.hostname.length() == 0) {
81     System.err.println(cliHelper.getMissingOptionMessage("hostname"));
82     System.err.flush();
83     System.out.println(cliHelper.getHelpMessage());
84     System.out.flush();
85     System.exit(1);
86 }
87
88 String portS = cliHelper.getValue("port");
89 status.port = portS != null && portS.length() != 0 ? Integer.parseInt(portS) :
    → DEFAULT_PORT;
90
91 // Init status
92 status.credentials = Credentials.create(status.privateKey);
93 status.catalogManager = new CatalogManager(status.credentials, catalogAddress);
94
95 // Create server
96 HttpServer server = HttpServer.create(new InetSocketAddress(status.port), 0);
97 System.out.println("Server running on port " + status.port + ".\n");
98
99 // set handlers
100 server.createContext("/deploy", CobraHttpHelper.newHandler(Main::deploy, status));
101 server.createContext("/access", CobraHttpHelper.newHandler(Main::access, status));
102
103 // start server
104 server.setExecutor(null); // creates a default executor
105 server.start();
106 }
107
108

```

```

109  /**
110   * Handler for the /deploy url.
111   * @param request a POST request with JSON encoded data containing:
112   *               privateKey of the author,
113   *               name of the content,
114   *               genre of the content (can be null),
115   *               price of the content (if null is set to 0),
116   *               port on which is running the server socket that uploads the file.
117   * @param status the server {@link Status object}.
118   */
119  private static void deploy(HttpExchange request, Status status) {
120      // get parameters
121      Map<String, String> parameters = CobraHttpHelper.parsePOST(request);
122
123      if (!status.privateKey.equals(parameters.get("privateKey"))) {
124          CobraHttpHelper.sendResponse(request, "Only the author server owner can perform
125              ↳ this action." +
126                  "You must login with the same private key of the server.", 403);
127          return;
128      }
129
130      String name = parameters.get("name");
131      if (name == null) {
132          System.err.println("ERROR: name not specified");
133          CobraHttpHelper.sendResponse(request, "ERROR: name not specified.", 400);
134          return;
135      }
136
137      String genre = parameters.get("genre");
138
139      String priceS = parameters.get("price");
140      BigInteger price;
141      try {
142          price = new BigInteger(priceS.length() != 0 ? priceS : "0");
143      } catch (NumberFormatException e) {
144          e.printStackTrace();
145          CobraHttpHelper.sendResponse(request, "ERROR: Invalid price.\n" + e.getMessage(),
146              ↳ 400);
147          return;
148      }
149
150      String hostname = request.getRemoteAddress().getHostName();
151
152      String portS = parameters.get("port");
153      int port;
154      try {
155          port = Integer.parseInt(portS);
156      } catch (NumberFormatException e) {
157          e.printStackTrace();
158          CobraHttpHelper.sendResponse(request, "ERROR: Invalid port number.\n" +
159              ↳ e.getMessage(), 400);
160          return;
161      }
162
163      String filename = parameters.get("filename");
164      if (filename == null) {
165          System.err.println("ERROR: filename not specified");
166          CobraHttpHelper.sendResponse(request, "ERROR: filename not specified.", 400);
167          return;
168      }

```

```

165     }
166
167     // deploy the content
168     String address;
169     try {
170         ContentManager contentManager = new ContentManager(status.credentials,
171             status.catalogManager.getAddress(), name, genre, price, status.hostname,
172             ↪ status.port);
173         address = contentManager.getAddress();
174     } catch (Exception e) {
175         e.printStackTrace();
176         CobraHttpHelper.sendResponse(request, e.getMessage(), 400);
177         return;
178     }
179
180     // download the file
181     File file = new File(CONTENT_FILE_PATH + address + filename);
182     //noinspection ResultOfMethodCallIgnored
183     file.getParentFile().mkdirs();
184     FileExchange.receiveFile(file, hostname, port);
185
186     // send the response
187     CobraHttpHelper.sendResponse(request, address);
188 }
189
190 /**
191  * Handler for the /access url.
192  * @param request a POST request with JSON encoded data containing:
193  *     privateKey of the author,
194  *     name of the content,
195  *     genre of the content (can be null),
196  *     price of the content (if null is set to 0).
197  * @param status the server {@link Status object}.
198  */
199 private static void access(HttpExchange request, Status status) {
200     // get parameters
201     Map<String, String> parameters = CobraHttpHelper.parsePOST(request);
202
203     String address = parameters.get("address");
204     if (address == null) {
205         CobraHttpHelper.sendResponse(request, "ERROR: content address not specified.",
206             ↪ 400);
207         return;
208     }
209
210     String userPrivateKey = parameters.get("privateKey");
211     if (userPrivateKey == null) {
212         CobraHttpHelper.sendResponse(request, "ERROR: user private key not specified.",
213             ↪ 400);
214         return;
215     }
216
217     Credentials credentials = Credentials.create(userPrivateKey);
218     String user = credentials.getAddress();
219
220     if (!status.catalogManager.hasAccess(address, user)) {
221         CobraHttpHelper.sendResponse(request, "ERROR: you don't have access to this
222             ↪ content.", 400);
223         return;
224     }

```

```

220
221 // open the socket for the file
222 ServerSocketChannel serverSocketChannel = FileExchange.openFileSocket();
223 if (serverSocketChannel == null) {
224     CobraHttpHelper.sendResponse(request, "ERROR: cannot open the server socket.",
225         ↪ 500);
226     return;
227 }
228
229 // pick the file
230 File[] files = new File(CONTENT_FILE_PATH).listFiles();
231 if (files == null) {
232     CobraHttpHelper.sendResponse(request, "ERROR: there is no file for this content.",
233         ↪ 500);
234     return;
235 }
236 File file = null;
237 for (File f : files)
238     if (f.isFile() && f.getName().contains(address)) {
239         file = f;
240         break;
241     }
242 if (file == null) {
243     CobraHttpHelper.sendResponse(request, "ERROR: there is no file for this content.",
244         ↪ 500);
245     return;
246 }
247 String filename = file.getName().replace(address, "");
248
249 int port = serverSocketChannel.socket().getLocalPort();
250 FileExchange.startFileSender(serverSocketChannel, file,
251     () -> System.out.println("User " + user + " has received all the content " +
252         ↪ address + "."));
253
254 // consume the content
255 ContentManager contentManager = new ContentManager(credentials, address);
256 if (!contentManager.consumeContent()) {
257     CobraHttpHelper.sendResponse(request, "ERROR: cannot consume content.", 500);
258     return;
259 }
260
261 // communicate the port number and the filename
262 Map<String, String> response = new HashMap<>();
263 response.put("port", Integer.toString(port));
264 response.put("filename", filename);
265 CobraHttpHelper.sendResponse(request, CobraHttpHelper.jsonifyParameters(response));
266 }
267
268 }

```

2.2 DAPP/connections

API.java

```
1 package com.aldodaquino.cobra.connections;
2
3 /**
4  * Interface for the author server API. Specify the urls of the handlers.
5  * Check Main class in {@link com.aldodaquino.cobra.authorserver}
6  * @author Aldo D'Aquino.
7  * @version 1.0.
8  */
9 public interface API {
10
11     String DEPLOY_API_PATH = "/deploy";
12     String ACCESS_API_PATH = "/access";
13
14 }
```

CobraHttpHelper.java

```
1 package com.aldodaquino.cobra.connections;
2
3 import com.aldodaquino.javautils.HttpHelper;
4 import com.sun.net.httpserver.HttpExchange;
5 import com.sun.net.httpserver.HttpHandler;
6
7 import java.util.function.BiConsumer;
8
9 /**
10  * Contains method that help to make http request.
11  * Works with JSON body for POST request and query-style GET parameters.
12  * @author Aldo D'Aquino.
13  * @version 1.0.
14  */
15 public class CobraHttpHelper extends HttpHelper {
16
17     private static class HttpRequestHandler implements HttpHandler {
18         final BiConsumer<HttpExchange, Status> consumer;
19         final Status status;
20         HttpRequestHandler(BiConsumer<HttpExchange, Status> consumer, Status status) {
21             this.consumer = consumer;
22             this.status = status;
23         }
24         @Override
25         public void handle(HttpExchange request) {
26             consumer.accept(request, status);
27         }
28     }
29
30     /**
31      * Return new HttpHandler
32      * @param consumer a function to be called when a new request arrive.
33      * The consumer has to accept the HttpExchange request and the Status.
34      * @param status to be passed to the consumer.
35      * @return HttpHandler, the handler.
36      */
37     public static HttpHandler newHandler(BiConsumer<HttpExchange, Status> consumer, Status
38 ↪ status) {
```



```
38         return new HttpRequestHandler(consumer, status);
39     }
40
41 }
```

Status.java

```
1  package com.aldodaquino.cobra.connections;
2
3  import com.aldodaquino.cobra.main.CatalogManager;
4  import org.web3j.crypto.Credentials;
5
6  /**
7   * Defines the status for the author-server.
8   * @author Aldo D'Aquino.
9   * @version 1.0.
10  */
11 public class Status {
12     public String privateKey;
13     public Credentials credentials;
14     public CatalogManager catalogManager;
15     public String hostname;
16     public int port;
17 }
```

2.3 DAPP/contracts

CatalogManager.java

```
1 package com.aldodaquino.cobra.main;
2
3 import com.aldodaquino.cobra.contracts.CatalogContract;
4 import org.web3j.crypto.Credentials;
5 import org.web3j.protocol.core.DefaultBlockParameterName;
6 import org.web3j.tuples.generated.*;
7
8 import java.math.BigInteger;
9 import java.util.ArrayList;
10 import java.util.HashMap;
11 import java.util.List;
12 import java.util.Map;
13 import java.util.function.BiConsumer;
14
15 /**
16  * An higher level Catalog Manager.
17  * Contains methods that call the methods in the {@link CatalogContract} generated by Web3j and
18  * → parse and aggregate the
19  * result in a more comfortable representation.
20  * @author Aldo D'Aquino.
21  * @version 1.0.
22  */
23 public class CatalogManager extends ContractManager {
24
25     private final CatalogContract catalog;
26
27     // event callbacks
28     private final List<BiConsumer<String, String>> newContentAvailableBiConsumers = new
29         ↳ ArrayList<>();
30     private final Map<String, List<BiConsumer<String, String>>> accessGrantedMap = new
31         ↳ HashMap<>();
32     private final Map<String, List<Runnable>> becomesPremiumMap = new HashMap<>();
33     private final List<BiConsumer<String, String>> feedbackAvailableBiConsumer = new
34         ↳ ArrayList<>();
35     private final Map<String, List<BiConsumer<String, String>>> paymentAvailableMap = new
36         ↳ HashMap<>();
37     private final List<Runnable> catalogClosedRunnables = new ArrayList<>();
38
39     /*
40     * CONSTRUCTORS
41     */
42
43     /**
44     * Deploy and manage a new catalog contract.
45     * @param credentials your account credentials.
46     */
47     public CatalogManager(Credentials credentials) {
48         super(credentials);
49         catalog = (CatalogContract) deploy(CatalogContract.class);
50         listenForEvents();
51     }
52
53     /**
54     * Load and manage an existent catalog contract.
55     * @param credentials your account credentials.
56     * @param contractAddress the existent contract address on blockchain.
```

```

52     */
53     public CatalogManager(Credentials credentials, String contractAddress) {
54         super(credentials);
55         catalog = (CatalogContract) load(CatalogContract.class, contractAddress);
56         listenForEvents();
57     }
58
59     // auxiliary function
60     private void listenForEvents() {
61         catalog.newContentAvailableEventObservable(DefaultBlockParameterName.EARLIEST,
62             ↳ DefaultBlockParameterName.LATEST)
63             .subscribe(e -> {
64                 for (BiConsumer<String, String> biConsumer :
65                     ↳ newContentAvailableBiConsumers)
66                     biConsumer.accept(Utils.bytes32ToString(e.name), e.addr);
67             });
68
69         catalog.grantedAccessEventObservable(DefaultBlockParameterName.EARLIEST,
70             ↳ DefaultBlockParameterName.LATEST)
71             .subscribe(e -> {
72                 List<BiConsumer<String, String>> biConsumers =
73                     ↳ accessGrantedMap.get(e.user);
74                 if (biConsumers != null) {
75                     for (BiConsumer<String, String> biConsumer : biConsumers)
76                         biConsumer.accept(e.content, getTheName(e.content));
77                 }
78             });
79
80         catalog.becomesPremiumEventObservable(DefaultBlockParameterName.EARLIEST,
81             ↳ DefaultBlockParameterName.LATEST)
82             .subscribe(e -> {
83                 List<Runnable> runnables = becomesPremiumMap.get(e.user);
84                 if (runnables != null)
85                     for (Runnable runnable : runnables)
86                         runnable.run();
87             });
88
89         catalog.feedbackAvailableEventObservable(DefaultBlockParameterName.EARLIEST,
90             ↳ DefaultBlockParameterName.LATEST)
91             .subscribe(e -> {
92                 if (e.user.equals(credentials.getAddress())) {
93                     String name = "";
94                     try {
95                         name =
96                             ↳ Utils.bytes32ToString(catalog.getContentInfo(e.content).send().getValue1());
97                     } catch (Exception exception) {
98                         exception.printStackTrace();
99                     }
100                     for (BiConsumer<String, String> biConsumer :
101                         ↳ feedbackAvailableBiConsumer)
102                         biConsumer.accept(e.content, name);
103                 }
104             });
105
106         catalog.paymentAvailableEventObservable(DefaultBlockParameterName.EARLIEST,
107             ↳ DefaultBlockParameterName.LATEST)
108             .subscribe(e -> {
109                 List<BiConsumer<String, String>> biConsumers =
110                     ↳ paymentAvailableMap.get(e.content);

```

```

101         if (biConsumers != null) {
102             for (BiConsumer<String, String> biConsumer : biConsumers)
103                 biConsumer.accept(e.content, getName(e.content));
104         }
105     });
106
107     catalog.catalogClosedEventObservable(DefaultBlockParameterName.EARLIEST,
108     ↪ DefaultBlockParameterName.LATEST)
109         .subscribe(e -> {
110             for (Runnable runnable : catalogClosedRunnables)
111                 runnable.run();
112         });
113
114     // auxiliary function
115     private String getName(String contentAddress) {
116         try {
117             return
118                 ↪ Utils.bytes32ToString(catalog.getContentInfo(contentAddress).send().getValue1());
119         } catch (Exception exception) {
120             exception.printStackTrace();
121             return "";
122         }
123     }
124
125     /*
126     * CATALOG CONTRACT SPECIFIC METHODS
127     */
128
129     /* Events */
130
131     /**
132     * Subscribe a callback for new content available events.
133     * @param callback a BiConsumer of content name and address.
134     */
135     public void listenNewContentAvailable(BiConsumer<String, String> callback) {
136         newContentAvailableBiConsumers.add(callback);
137     }
138
139     /**
140     * Subscribe a callback for new content available events for a specified genre or author.
141     * @param authorOrGenre the genre or the author address for which listen to.
142     * @param callback a BiConsumer of content name and address.
143     */
144     public void listenNewContentAvailable(String authorOrGenre, BiConsumer<String, String>
145     ↪ callback) {
146         newContentAvailableBiConsumers.add((address, name) -> {
147             Content content = getContentInfo(address);
148             if (content.author.equals(authorOrGenre) || content.genre.equals(authorOrGenre))
149                 callback.accept(address, name);
150         });
151     }
152
153     /**
154     * Subscribe a callback for access granted events for this user.
155     * @param callback a BiConsumer of content address and content name.
156     */
157     public void listenAccessGranted(BiConsumer<String, String> callback) {
158         listenAccessGranted(credentials.getAddress(), callback);
159     }

```

```

157
158 /**
159  * Subscribe a callback for access granted events for the specified user.
160  * @param user the user for which be registered.
161  * @param callback a BiConsumer of content address and content name.
162  */
163 public void listenAccessGranted(String user, BiConsumer<String, String> callback) {
164     accessGrantedMap.putIfAbsent(user, new ArrayList<>());
165     accessGrantedMap.get(user).add(callback);
166 }
167
168 /**
169  * Subscribe a callback for becomes premium events for this user.
170  * @param callback a Runnable.
171  */
172 public void listenBecomesPremium(Runnable callback) {
173     listenBecomesPremium(credentials.getAddress(), callback);
174 }
175
176 /**
177  * Subscribe a callback for becomes premium events for this user.
178  * @param user the user for which be registered.
179  * @param callback a Runnable.
180  */
181 public void listenBecomesPremium(String user, Runnable callback) {
182     becomesPremiumMap.putIfAbsent(user, new ArrayList<>());
183     becomesPremiumMap.get(user).add(callback);
184 }
185
186 /**
187  * Subscribe a callback for feedback available events for this user.
188  * @param callback a BiConsumer of content address and content name.
189  */
190 public void listenFeedbackAvailable(BiConsumer<String, String> callback) {
191     feedbackAvailableBiConsumer.add(callback);
192 }
193
194 /**
195  * Subscribe a callback for payment available events for this user.
196  * @param content the content of which listen to.
197  * @param callback a BiConsumer of content address and content name.
198  */
199 public void listenPaymentAvailable(String content, BiConsumer<String, String> callback) {
200     paymentAvailableMap.putIfAbsent(content, new ArrayList<>());
201     paymentAvailableMap.get(content).add(callback);
202 }
203
204 /**
205  * Subscribe a callback for payment available events for this user.
206  * @param callback a Runnable.
207  */
208 public void listenCatalogClosed(Runnable callback) {
209     catalogClosedRunnables.add(callback);
210 }
211
212 /* Catalog interaction methods */
213
214 /**
215  * Check if the user has access to a content.

```

```

216     * @param address the content address.
217     * @return boolean if has access, false otherwise.
218     */
219     public boolean hasAccess(String address) {
220         return hasAccess(address, credentials.getAddress());
221     }
222
223     /**
224     * Check if the specified user has access to a content.
225     * @param address the content address.
226     * @param user the user address.
227     * @return boolean if has access, false otherwise.
228     */
229     public boolean hasAccess(String address, String user) {
230         try {
231             return catalog.hasAccess(user, address).send();
232         } catch (Exception e) {
233             e.printStackTrace();
234             return false;
235         }
236     }
237
238     /**
239     * Buy a content.
240     * @param address the content address.
241     * @param price the content price.
242     * @return a boolean representing the operation outcome.
243     */
244     public boolean buyContent(String address, BigInteger price) {
245         try {
246             return catalog.getContent(address, price).send().isStatusOK();
247         } catch (Exception e) {
248             e.printStackTrace();
249             return false;
250         }
251     }
252
253     /**
254     * Gift a content to another user.
255     * @param address the content address.
256     * @param user the user address.
257     * @param price the content price.
258     * @return a boolean representing the operation outcome.
259     */
260     public boolean giftContent(String address, String user, BigInteger price) {
261         try {
262             return catalog.giftContent(address, user, price).send().isStatusOK();
263         } catch (Exception e) {
264             e.printStackTrace();
265             return false;
266         }
267     }
268
269     /**
270     * Buy a premium subscription.
271     * @return a boolean representing the operation outcome.
272     */
273     public boolean buyPremium() {
274         try {

```

```

275         BigInteger premiumCost = catalog.premiumCost().send();
276         return catalog.buyPremium(premiumCost).send().isStatusOK();
277     } catch (Exception e) {
278         e.printStackTrace();
279         return false;
280     }
281 }
282
283 /**
284  * Gift a premium subscription to another user.
285  * @param user the user address.
286  * @return a boolean representing the operation outcome.
287  */
288 public boolean giftPremium(String user) {
289     try {
290         BigInteger contentCost = catalog.premiumCost().send();
291         return catalog.giftPremium(user, contentCost).send().isStatusOK();
292     } catch (Exception e) {
293         e.printStackTrace();
294         return false;
295     }
296 }
297
298 /**
299  * Return true if the user has an active premium subscription.
300  * @return a boolean representing the operation outcome.
301  */
302 public boolean isPremium() {
303     try {
304         return catalog.isPremium(credentials.getAddress()).send();
305     } catch (Exception e) {
306         e.printStackTrace();
307         return false;
308     }
309 }
310
311 /**
312  * Set rating for a content.
313  * @param content the content address.
314  * @param enjoy the rating for the enjoy category.
315  * @param valueForMoney the rating for the value for money category.
316  * @param contentMeaning the rating for the content meaning category.
317  */
318 public void vote(String content, int enjoy, int valueForMoney, int contentMeaning) {
319     try {
320         byte[] enjoyS = catalog.ratingCategories(new BigInteger("0")).send();
321         byte[] valueForMoneyS = catalog.ratingCategories(new BigInteger("1")).send();
322         byte[] contentMeaningS = catalog.ratingCategories(new BigInteger("2")).send();
323         catalog.leaveFeedback(content, enjoyS, new BigInteger(Integer.toString(enjoy)));
324         catalog.leaveFeedback(content, valueForMoneyS, new
325             ↪ BigInteger(Integer.toString(valueForMoney)));
326         catalog.leaveFeedback(content, contentMeaningS, new
327             ↪ BigInteger(Integer.toString(contentMeaning)));
328     } catch (Exception e) {
329         e.printStackTrace();
330     }
331 }
332
333 /** Getters for lists, statistics and charts */

```

```

332
333 /**
334  * Returns all the info and ratings of a content.
335  * @param address the content address.
336  * @return a list of Content objects.
337  */
338 public Content getContentInfo(String address) {
339     try {
340         Tuple5<byte[], String, byte[], BigInteger, BigInteger> info =
341             ↪ catalog.getContentInfo(address).send();
342         Tuple4<BigInteger, BigInteger, BigInteger, BigInteger> ratings =
343             ↪ catalog.getContentRatings(address).send();
344
345         return new Content(address, info.getValue1(), info.getValue2(), info.getValue3(),
346             ↪ info.getValue4(),
347                 info.getValue5(), ratings.getValue1(), ratings.getValue2(),
348                 ↪ ratings.getValue3(), ratings.getValue4());
349     } catch (Exception e) {
350         e.printStackTrace();
351         return null;
352     }
353 }
354
355 /**
356  * Returns a list of all contents in the Catalog.
357  * @return a list of Content objects.
358  */
359 public List<Content> getContentList() {
360     try {
361         Tuple2<List<byte[]>, List<String>> statistics = catalog.getContentList().send();
362         List<byte[]> names = statistics.getValue1();
363         List<String> addresses = statistics.getValue2();
364
365         List<Content> contents = new ArrayList<>();
366         for (int i = 0; i < names.size(); i++)
367             contents.add(new Content(addresses.get(i), names.get(i)));
368
369         return contents;
370     } catch (Exception e) {
371         e.printStackTrace();
372         return null;
373     }
374 }
375
376 /**
377  * Returns a list of all contents in the Catalog and its views.
378  * @return a list of Content objects.
379  */
380 public List<Content> getContentListWithViews() {
381     try {
382         Tuple3<List<byte[]>, List<String>, List<BigInteger>> statistics =
383             ↪ catalog.getStatistics().send();
384         List<byte[]> names = statistics.getValue1();
385         List<String> addresses = statistics.getValue2();
386         List<BigInteger> views = statistics.getValue3();
387
388         List<Content> contents = new ArrayList<>();
389         for (int i = 0; i < names.size(); i++)
390             contents.add(new Content(addresses.get(i), names.get(i), views.get(i)));
391     }

```



```

386         return contents;
387     } catch (Exception e) {
388         e.printStackTrace();
389         return null;
390     }
391 }
392
393
394 /**
395  * Return the list of all the content of a given author.
396  * @param author the authors address.
397  * @return a list of Content objects.
398  */
399 public List<Content> getAuthorContents(String author) {
400     ContentList contentList = new ContentList();
401     return contentList.getFilteredContentList(contentList.authors, author);
402 }
403
404 /**
405  * Return the n latest releases.
406  * @param n the number of item that you want in the list.
407  * @return List of Content objects with the latest n contents.
408  */
409 public List<Content> getNewContentList(int n) {
410     try {
411         // get the list
412         Tuple2<List<byte[]>, List<String>> res =
413             catalog.getNewContentList(new BigInteger(Integer.toString(n))).send();
414         List<byte[]> names = res.getValue1();
415         List<String> addresses = res.getValue2();
416
417         // parse the list in a String matrix
418         List<Content> contents = new ArrayList<>();
419         for (int i = 0; i < names.size(); i++)
420             contents.add(new Content(addresses.get(i), names.get(i)));
421         return contents;
422     } catch (Exception e) {
423         e.printStackTrace();
424         return null;
425     }
426 }
427
428 /**
429  * Return the latest release.
430  * @return String[] where the first element is the name of the content and the second is
431  * the address.
432  */
433 public Content getLatest() {
434     List<Content> contents = getNewContentList(1);
435     if (contents == null || contents.size() == 0) return null;
436     return contents.get(0);
437 }
438
439 /**
440  * Return the latest release for a genre.
441  * @param genre the chosen genre.
442  * @return String[] where the first element is the name of the content and the second is
443  * the address.

```

```

443 public Content getLatestByGenre(String genre) {
444     try {
445         Tuple2<byte[], String> res =
446             ↪ catalog.getLatestByGenre(Utils.stringToBytes32(genre)).send();
447         return new Content(res.getValue2(), res.getValue1());
448     } catch (Exception e) {
449         e.printStackTrace();
450         return null;
451     }
452 }
453 /**
454  * Return the latest release of an author.
455  * @param author the author address.
456  * @return String[] where the first element is the name of the content and the second is
457  ↪ the address.
458  */
459 public Content getLatestByAuthor(String author) {
460     try {
461         Tuple2<byte[], String> res = catalog.getLatestByAuthor(author).send();
462         return new Content(res.getValue2(), res.getValue1());
463     } catch (Exception e) {
464         e.printStackTrace();
465         return null;
466     }
467 }
468 /**
469  * Return the most popular content.
470  * @return Content.
471  */
472 public Content getMostPopular() {
473     try {
474         Tuple2<byte[], String> res = catalog.getMostPopular().send();
475         return new Content(res.getValue2(), res.getValue1());
476     } catch (Exception e) {
477         e.printStackTrace();
478         return null;
479     }
480 }
481 /**
482  * Return the most popular content for a genre.
483  * @param genre the chosen genre.
484  * @return String[] where the first element is the name of the content and the second is
485  ↪ the address.
486  */
487 public Content getMostPopularByGenre(String genre) {
488     try {
489         Tuple2<byte[], String> res =
490             ↪ catalog.getMostPopularByGenre(Utils.stringToBytes32(genre)).send();
491         return new Content(res.getValue2(), res.getValue1());
492     } catch (Exception e) {
493         e.printStackTrace();
494         return null;
495     }
496 }
497 /**

```

```

498     * Return the most popular content of an author.
499     * @param author the author address.
500     * @return String[] where the first element is the name of the content and the second is
↪ the address.
501     */
502     public Content getMostPopularByAuthor(String author) {
503         try {
504             Tuple2<byte[], String> res = catalog.getMostPopularByAuthor(author).send();
505             return new Content(res.getValue2(), res.getValue1());
506         } catch (Exception e) {
507             e.printStackTrace();
508             return null;
509         }
510     }
511
512     /**
513     * Return the highest rated content.
514     * @param category the category name for which you want to know the rating.
515     * @return Content.
516     */
517     public Content getMostRated(String category) {
518         try {
519             Tuple2<byte[], String> res =
↪ catalog.getMostRated(Utils.stringToBytes32(category)).send();
520             return new Content(res.getValue2(), res.getValue1());
521         } catch (Exception e) {
522             e.printStackTrace();
523             return null;
524         }
525     }
526
527     /**
528     * Return the highest rated content for a genre.
529     * @param genre the chosen genre.
530     * @param category the category name for which you want to know the rating.
531     * @return Content.
532     */
533     public Content getMostRatedByGenre(String genre, String category) {
534         try {
535             Tuple2<byte[], String> res =
↪ catalog.getMostRatedByGenre(Utils.stringToBytes32(genre),
536                               Utils.stringToBytes32(category)).send();
537             return new Content(res.getValue2(), res.getValue1());
538         } catch (Exception e) {
539             e.printStackTrace();
540             return null;
541         }
542     }
543
544     /**
545     * Return the highest rated content of an author.
546     * @param author the author address.
547     * @param category the category name for which you want to know the rating.
548     * @return Content.
549     */
550     public Content getMostRatedByAuthor(String author, String category) {
551         try {
552             Tuple2<byte[], String> res = catalog.getMostRatedByAuthor(author,
↪ Utils.stringToBytes32(category)).send();

```

```

553         return new Content(res.getValue2(), res.getValue1());
554     } catch (Exception e) {
555         e.printStackTrace();
556         return null;
557     }
558 }
559
560 /* Authors method */
561
562 /**
563  * Collect the payout for a content. Can be called only on content of the current user.
564  * @param address the address of the content.
565  * @return a BigInteger with the withdrawn amount, 0 otherwise (for example if the
↪ specified content is not owned by
566  * the user that have done the login).
567 */
568 public BigInteger withdraw(String address) {
569     try {
570         BigInteger amount = catalog.payoutAvailable(address).send();
571         if (!amount.equals(BigInteger.ZERO))
572             catalog.collectPayout(address).send();
573         return amount;
574     } catch (Exception e) {
575         e.printStackTrace();
576         return BigInteger.ZERO;
577     }
578 }
579
580
581 /* Auxiliary class */
582 private class ContentList {
583
584     List<String> addresses;
585     List<byte[]> names;
586     List<String> authors;
587     List<byte[]> genres;
588     List<BigInteger> prices;
589     List<BigInteger> views;
590     List<BigInteger> averageRatings;
591     List<BigInteger> enjoyRatings;
592     List<BigInteger> priceFairnessRatings;
593     List<BigInteger> contentMeaningRatings;
594
595     ContentList() {
596         try {
597             // Query the CatalogContract for the list
598             Tuple6<List<String>, List<byte[]>, List<String>, List<byte[]>,
↪ List<BigInteger>, List<BigInteger>>
599                 fullContentList = catalog.getFullContentList().send();
600             Tuple5<List<String>, List<BigInteger>, List<BigInteger>, List<BigInteger>,
↪ List<BigInteger>>
601                 ratingsList = catalog.getRatingsList().send();
602
603             // Parse parameters
604             addresses = fullContentList.getValue1();
605             names = fullContentList.getValue2();
606             authors = fullContentList.getValue3();
607             genres = fullContentList.getValue4();
608             prices = fullContentList.getValue5();

```

```

609         views = fullContentList.getValue6();
610         averageRatings = ratingsList.getValue2();
611         enjoyRatings = ratingsList.getValue3();
612         priceFairnessRatings = ratingsList.getValue4();
613         contentMeaningRatings = ratingsList.getValue5();
614     } catch (Exception e) {
615         e.printStackTrace();
616     }
617 }
618
619 /**
620  * Returns a list of all contents that has the parameter where equals to value.
621  * The list is not filtered if where is null.
622  * @param filterBy a list of this class that can be addresses, names, authors, genres,
↪ prices or views.
623  * @param filterValue the value that filterBy must have.
624  * @param <T> the type of the elements in one of this List: addresses, names, authors,
↪ genres, prices or views.
625  * It can be String, BigInteger or byte[].
626  * @return a list of Content objects.
627  */
628 <T> List<Content> getFilteredContentList(List<T> filterBy, T filterValue) {
629     // Build an usable list
630     List<Content> contentList = new ArrayList<>();
631     for (int i = 0; i < addresses.size(); i++)
632         // if the where list is null do not filter
633         if (filterBy == null || filterBy.get(i).equals(filterValue))
634             contentList.add(new Content(
635                 addresses.get(i),
636                 names.get(i),
637                 authors.get(i),
638                 genres.get(i),
639                 prices.get(i),
640                 views.get(i),
641                 averageRatings.get(i),
642                 enjoyRatings.get(i),
643                 priceFairnessRatings.get(i),
644                 contentMeaningRatings.get(i)
645             ));
646     return contentList;
647 }
648 }
649
650 }

```

Content.java

```

1  package com.aldodaquino.cobra.main;
2
3  import java.math.BigInteger;
4
5  /**
6   * Defines a content object.
7   * @author Aldo D'Aquino.
8   * @version 1.0.
9   */
10 public class Content {
11

```

```

12 public final String address;           // required
13 public final String name;             // required
14 public final String author;           // can be null
15 public final String genre;            // can be null
16 public final BigInteger price;         // can be null
17 public final BigInteger views;         // can be null
18 public final int averageRating;        // can be -1
19 public final int enjoy;                // can be -1
20 public final int priceFairness;        // can be -1
21 public final int contentMeaning;       // can be -1
22
23 Content(String address, byte[] name) {
24     this.address = address;
25     this.name = Utils.bytes32ToString(name);
26
27     this.views = null;
28     this.author = null;
29     this.genre = null;
30     this.price = null;
31     this.averageRating = -1;
32     this.enjoy = -1;
33     this.priceFairness = -1;
34     this.contentMeaning = -1;
35 }
36
37 Content(String address, byte[] name, BigInteger views) {
38     this.address = address;
39     this.name = Utils.bytes32ToString(name);
40     this.views = views;
41
42     this.author = null;
43     this.genre = null;
44     this.price = null;
45     this.averageRating = -1;
46     this.enjoy = -1;
47     this.priceFairness = -1;
48     this.contentMeaning = -1;
49 }
50
51 Content(String address, byte[] name, String author, byte[] genre, BigInteger price,
52 ↪     BigInteger views,
53     ↪     BigInteger averageRating, BigInteger enjoy, BigInteger priceFairness, BigInteger
54     ↪     contentMeaning) {
55     this.address = address;
56     this.name = Utils.bytes32ToString(name);
57     this.author = author;
58     this.genre = Utils.bytes32ToString(genre);
59     this.price = price;
60     this.views = views;
61     this.averageRating = averageRating.intValue();
62     this.enjoy = enjoy.intValue();
63     this.priceFairness = priceFairness.intValue();
64     this.contentMeaning = contentMeaning.intValue();
65 }

```

ContentManager.java

```

1  package com.aldodaquino.cobra.main;
2
3  import com.aldodaquino.cobra.contracts.DAPPContentManagementContract;
4  import org.web3j.crypto.Credentials;
5  import org.web3j.protocol.core.DefaultBlockParameterName;
6
7  import java.math.BigInteger;
8  import java.util.ArrayList;
9  import java.util.List;
10
11  /**
12   * An higher level Content Manager.
13   * Contains methods that call the methods in the {@link DAPPContentManagementContract}
14   * → generated by Web3j and parse and
15   * aggregate the result in a more comfortable representation.
16   * @author Aldo D'Aquino.
17   * @version 1.0.
18   */
19  public class ContentManager extends ContractManager {
20
21      private final DAPPContentManagementContract content;
22
23      private final List<Runnable> contentPublishedRunnables = new ArrayList<>();
24
25      /**
26       * Deploy and manage a new content manager contract.
27       * @param credentials your account credentials.
28       * @param catalogAddress the address of the catalog.
29       * @param name the name you want to assign to this content.
30       * @param genre the genre you want to assign to this content.
31       * @param price the price you want to assign to this content.
32       * @param hostname the hostname of the author server.
33       * @param port the port on which is running the author server.
34       * @throws Exception if there is errors when deploying the Content contract.
35       */
36      public ContentManager(Credentials credentials, String catalogAddress, String name, String
37          → genre, BigInteger price,
38          String hostname, int port)
39          throws Exception {
40          super(credentials);
41          content = (DAPPContentManagementContract) deploy(DAPPContentManagementContract.class);
42          content.setName(Utils.stringToBytes32(name)).send();
43          content.setGenre(Utils.stringToBytes32(genre)).send();
44          content.setPrice(price).send();
45          content.setHostname(Utils.stringToBytes32(hostname)).send();
46          content.setPort(new BigInteger(Integer.toString(port))).send();
47          content.publish(catalogAddress).send();
48
49          content.contentPublishedEventObservable(DefaultBlockParameterName.EARLIEST,
50          → DefaultBlockParameterName.LATEST)
51              .subscribe(e -> {
52                  for (Runnable runnable : contentPublishedRunnables)
53                      runnable.run();
54              });
55      }
56
57      /**
58       * Load and manage an existent content manager contract.
59       * @param credentials your account credentials.

```

```

57     * @param contractAddress the existent contract address on blockchain.
58     */
59     public ContentManager(Credentials credentials, String contractAddress) {
60         super(credentials);
61         content = (DAPPContentManagementContract) load(DAPPContentManagementContract.class,
62             ↪ contractAddress);
63     }
64
65     /**
66     * Subscribe a callback for content published events.
67     * @param callback a Runnable.
68     */
69     public void listenContentPublished(Runnable callback) {
70         contentPublishedRunnables.add(callback);
71     }
72
73     /**
74     * Consume a bought content.
75     * Note that the content is consumed by the user that owns the credentials passed to the
76     ↪ constructor.
77     * @return a boolean representing the operation outcome.
78     */
79     public boolean consumeContent() {
80         try {
81             return content.consumeContent().send().isStatusOK();
82         } catch (Exception e) {
83             e.printStackTrace();
84             return false;
85         }
86     }
87
88     /**
89     * Returns the hostname of the author-server.
90     * @return String the hostname.
91     */
92     public String getHostname() {
93         try {
94             return Utils.bytes32ToString(content.hostname().send());
95         } catch (Exception e) {
96             e.printStackTrace();
97             return null;
98         }
99     }
100
101     /**
102     * Returns the port number of the author-server.
103     * @return the int number of the port.
104     */
105     public int getPort() {
106         try {
107             return content.port().send().intValue();
108         } catch (Exception e) {
109             e.printStackTrace();
110             return 0;
111         }
112     }

```


ContractManager.java

```

1  package com.aldodaquino.cobra.main;
2
3  import org.web3j.crypto.Credentials;
4  import org.web3j.protocol.Web3j;
5  import org.web3j.protocol.core.RemoteCall;
6  import org.web3j.protocol.http.HttpService;
7  import org.web3j.tx.Contract;
8
9  import java.io.IOException;
10 import java.lang.reflect.InvocationTargetException;
11 import java.lang.reflect.Method;
12 import java.math.BigInteger;
13
14 /**
15  * An higher level Contract Manager.
16  * Contains basic methods of all the contracts, such as the deploy, load and suicide function,
17  * ↪ but also the get owner
18  * and get address functions.
19  * @author Aldo D'Aquino.
20  * @version 1.0.
21  */
22 class ContractManager {
23     private Web3j web3;
24     private BigInteger gasPrice;
25     private BigInteger gasLimit;
26     Credentials credentials;
27
28     private Contract contract;
29     private Class<? extends Contract> contractClass;
30     private String owner;
31
32     /*
33      * CONSTRUCTORS
34      */
35
36     /**
37      * Save credentials, connect to web3 and save the gas information.
38      * @param credentials your account credentials.
39      */
40     ContractManager(Credentials credentials) {
41         if (credentials == null) throw new IllegalArgumentException("Credentials cannot be
42         ↪ null.");
43         // save credentials
44         this.credentials = credentials;
45         // connect to web3
46         web3 = Web3j.build(new HttpService()); // defaults to http://localhost:8545/
47         try {
48             System.out.println("Web3 protocol version: " +
49             ↪ web3.ethProtocolVersion().send().getProtocolVersion());
50         } catch (IOException e) {
51             e.printStackTrace();
52         }
53         // get gas information
54         gasPrice = Utils.getGasPrice(web3);
55         gasLimit = Utils.getGasLimit(web3);
56         Utils.getBalance(web3, credentials.getAddress());

```

```

55     }
56
57     /**
58      * Deploy a new contract of class contractClass and return the contractClass instance.
59      * @param contractClass the class of the contract that you want deploy.
60      * @return the deployed contract as contractClass instance.
61      */
62     Contract deploy(Class<? extends Contract> contractClass) {
63         this.contractClass = contractClass;
64         try {
65             Object[] params = {web3, credentials, gasPrice, gasLimit};
66             Class[] paramsTypes = {Web3j.class, Credentials.class, BigInteger.class,
67                                     ↳ BigInteger.class};
68             Method deploy = contractClass.getMethod("deploy", paramsTypes);
69             contract = (Contract) ((RemoteCall) deploy.invoke(null, params)).send();
70             owner = credentials.getAddress(); // who deploy the contract is the owner
71         } catch (NoSuchMethodException | IllegalAccessException | InvocationTargetException e)
72             ↳ {
73             System.err.println("ERROR while deploying " + contractClass + ".");
74             e.printStackTrace();
75         } catch (Exception e) {
76             System.err.println("Got Web3j error while deploying " + contractClass + ".");
77             // I want to end the program if the exception occur,
78             // but I don't want to have to manage this exception that should not be thrown
79             throw new RuntimeException(e);
80         }
81         return contract;
82     }
83
84     /**
85      * Load an existent contract of class contractClass and return the contractClass instance.
86      * @param contractClass the class of the contract that you want to load.
87      * @param contractAddress the address of the contract.
88      * @return the loaded contract as contractClass instance.
89      */
90     Contract load(Class<? extends Contract> contractClass, String contractAddress) {
91         this.contractClass = contractClass;
92         Object[] params = {contractAddress, web3, credentials, gasPrice, gasLimit};
93         Class[] paramsTypes = {String.class, Web3j.class, Credentials.class, BigInteger.class,
94                                 ↳ BigInteger.class};
95         try {
96             // Load contract
97             Method load = contractClass.getMethod("load", paramsTypes);
98             contract = (Contract) load.invoke(null, params);
99             // Get the contract owner
100             Method owner = contractClass.getMethod("owner");
101             try {
102                 this.owner = (String) ((RemoteCall) owner.invoke(contract)).send();
103             } catch (NullPointerException e) {
104                 System.err.println("ERROR while loading " + contractClass +
105                                     " ". Contract " + contractAddress + " may not exists.");
106                 e.printStackTrace();
107             }
108         } catch (NoSuchMethodException | IllegalAccessException | InvocationTargetException e)
109             ↳ {
110             System.err.println("ERROR while loading " + contractClass + ".");
111             e.printStackTrace();
112         } catch (Exception e) {
113             System.err.println("Got Web3j error while trying to get the contract owner.");

```

```

110         e.printStackTrace();
111     }
112     return contract;
113 }
114
115 /**
116  * Returns the contract address.
117  * @return a string containing the contract address.
118  */
119 public String getAddress() {
120     return contract.getContractAddress();
121 }
122
123 /**
124  * Returns the contract owner.
125  * @return a string containing the contract owner.
126  */
127 public String getOwner() {
128     return owner;
129 }
130
131 /**
132  * Call the suicide function on the contract.
133  * @return true if the contract suicide has been committed, false in case of errors.
134  */
135 public boolean suicide() {
136     try {
137         Method suicide = contractClass.getMethod("_suicide");
138         contract = (Contract) ((RemoteCall) suicide.invoke(null)).send();
139         return true;
140     } catch (Exception e) {
141         System.err.println("Got Web3j error while try to get the contract owner.");
142         e.printStackTrace();
143         return false;
144     }
145 }
146
147 }

```

Utils.java

```

1  package com.aldodaquino.cobra.main;
2
3  import org.web3j.protocol.Web3j;
4  import org.web3j.protocol.core.DefaultBlockParameter;
5  import org.web3j.protocol.core.DefaultBlockParameterName;
6  import org.web3j.protocol.core.Ethereum;
7  import org.web3j.protocol.core.methods.response.EthBlock;
8
9  import java.io.IOException;
10 import java.math.BigInteger;
11 import java.nio.charset.StandardCharsets;
12 import java.util.Arrays;
13
14 /**
15  * Some utilities for the blockchain.
16  * Contains method to get gas information and to convert bytes32 to Strings and vice-versa.
17  * @author Aldo D'Aquino.

```

```

18  * @version 1.0.
19  */
20  class Utils {
21
22      private static final String BLOCK_GAS_LIMIT = "5000000";
23
24      /**
25       * Return the average gas price.
26       * @param web3 a web3j instance.
27       * @return a BigInteger of the gas price, 0 in case of error.
28       */
29      static BigInteger getGasPrice(Web3j web3) {
30          BigInteger gasPrice;
31          try {
32              gasPrice = web3.ethGasPrice().send().getGasPrice();
33          } catch (IOException e) {
34              System.err.println("Cannot get the gas limit.");
35              e.printStackTrace();
36              gasPrice = BigInteger.ZERO;
37          }
38          System.out.println("Gas price: " + gasPrice);
39          return gasPrice;
40      }
41
42      /**
43       * Return the maximum gas limit that we can use in a transaction.
44       * @param web3 a web3j instance.
45       * @return a BigInteger of the gas limit, 0 in case of error.
46       */
47      static BigInteger getGasLimit(Web3j web3) {
48          BigInteger gasLimit;
49          try {
50              EthBlock.Block block =
51                  web3.ethGetBlockByNumber(DefaultBlockParameterName.LATEST,
52                      true).send().getBlock();
53              if (block != null) {
54                  System.out.println("Latest block number: " + block.getNumber());
55                  gasLimit = block.getGasLimit();
56              }
57              else gasLimit = new BigInteger(BLOCK_GAS_LIMIT);
58          } catch (IOException e) {
59              System.err.println("Cannot get the gas limit.");
60              e.printStackTrace();
61              gasLimit = BigInteger.ZERO;
62          }
63          System.out.println("Block gas limit: " + gasLimit);
64          return gasLimit;
65      }
66
67      static BigInteger getBalance(Web3j web3, String address) {
68          BigInteger balance;
69          try {
70              balance = web3.ethGetBalance(address, DefaultBlockParameterName.LATEST).send()
71                  .getBalance();
72          } catch (IOException e) {
73              System.err.println("Cannot get the account balance.");
74              e.printStackTrace();
75              balance = BigInteger.ZERO;
76          }
77      }

```

```

76         System.out.println("Account balance: " + balance);
77         return balance;
78     }
79
80     /**
81      * Convert a bytes32 in a String.
82      * @param bytes32 the byte[].
83      * @return the String.
84      */
85     static String bytes32ToString(byte[] bytes32) {
86         int i = bytes32.length - 1;
87         while (i >= 0 && bytes32[i] == 0) i--;
88         bytes32 = Arrays.copyOf(bytes32, i + 1);
89         return new String(bytes32, StandardCharsets.UTF_8);
90     }
91
92     /**
93      * Convert a String in a bytes32.
94      * @param string the String.
95      * @return the byte[].
96      */
97     static byte[] stringToBytes32(String string) {
98         byte[] byte32 = new byte[32];
99         if (string != null) {
100             byte[] bytes = string.getBytes();
101             System.arraycopy(bytes, 0, byte32, 0, bytes.length);
102         }
103         return byte32;
104     }
105
106 }

```

2.4 DAPP/gui

Main.java

```
1  package com.aldodaquino.cobra.gui;
2
3  import com.aldodaquino.cobra.gui.constants.Strings;
4  import com.aldodaquino.cobra.gui.panels.AuthorPanel;
5  import com.aldodaquino.cobra.gui.panels.CustomerPanel;
6  import com.aldodaquino.cobra.gui.panels.StarterPanel;
7
8  import javax.swing.*;
9
10 /**
11  * The GUI Main. Starts the GUI with the {@link StarterPanel}.
12  * @author Aldo D'Aquino.
13  * @version 1.0.
14  */
15 public class Main {
16
17     private static JFrame window;
18
19     /**
20      * Main method.
21      * @param args an empty array, no parameters are required.
22      */
23     public static void main(String[] args) {
24
25         // Create the starter panel
26         JPanel starterPanel = new StarterPanel(Main::showMainPanel);
27
28         // Create the window
29         window = Utils.newWindow(Strings.appName, starterPanel, true);
30         window.setMinimumSize(starterPanel.getMinimumSize());
31
32     }
33
34     private static void setContent(JPanel replacement) {
35         window.setContentPane(replacement);
36         window.revalidate();
37         window.repaint();
38         window.pack();
39         window.setLocationRelativeTo(null);
40         window.setMinimumSize(replacement.getMinimumSize());
41     }
42
43     private static void showMainPanel(Status status) {
44         JPanel newPanel;
45
46         switch (status.getRole()) {
47             case (Status.ROLE_CUSTOMER):
48                 newPanel = new CustomerPanel(status);
49                 break;
50             case (Status.ROLE_AUTHOR):
51                 newPanel = new AuthorPanel(status);
52                 break;
53             default:
54                 throw new IllegalArgumentException("Status role property has an invalid value.
55                 ↪      " +
56                     "Should be one of the Status.ROLE_X constants.");
```

```

56     }
57     setContent(newPanel);
58 }
59 }

```

Status.java

```

1  package com.aldodaquino.cobra.gui;
2
3  import com.aldodaquino.cobra.main.CatalogManager;
4  import org.web3j.crypto.Credentials;
5
6  import javax.naming.OperationNotSupportedException;
7
8  /**
9   * The Status class. A Status object is generated in the {@link
10   ↪ com.aldodaquino.cobra.gui.panels.StarterPanel}.
11   * Contains all the information about the user and the catalog and is passed through the
12   ↪ classes.
13   * @author Aldo D'Aquino.
14   * @version 1.0.
15   */
16 public class Status {
17
18     public static final int ROLE_CUSTOMER = 0;
19     public static final int ROLE_AUTHOR = 1;
20
21     private String privateKey;
22     public Credentials credentials;
23     private CatalogManager catalogManager;
24     private int role;
25
26     /**
27      * Given a private key generates and stores the credentials for this user.
28      * @param privateKey the user's private key.
29      * @throws OperationNotSupportedException if the user is already logged in.
30      */
31     public void login (String privateKey) throws OperationNotSupportedException {
32         this.privateKey = privateKey;
33         if (privateKey == null || privateKey.length() == 0) throw new
34             ↪ IllegalArgumentException("Empty private key");
35         if (credentials != null)
36             throw new OperationNotSupportedException("Already logged in as " +
37                 ↪ credentials.getAddress() + ".");
38         credentials = Credentials.create(privateKey);
39     }
40
41     /**
42      * Connects to an existent catalog and stores it.
43      * @param catalogAddress the catalog address.
44      * @throws OperationNotSupportedException if the user is not logged in.
45      */
46     public void connectCatalog(String catalogAddress) throws OperationNotSupportedException {
47         if (catalogAddress == null || catalogAddress.length() == 0)
48             throw new IllegalArgumentException("Empty catalog address");
49         if (credentials == null)
50             throw new OperationNotSupportedException("You must be logged in to connect to a
51                 ↪ catalog.");
52     }
53 }

```

```

47         catalogManager = new CatalogManager(credentials, catalogAddress);
48     }
49
50     /**
51      * Disconnects from the catalog.
52      */
53     public void disconnectCatalog() {
54         catalogManager = null;
55     }
56
57     /**
58      * Deploys a new catalog with the user credentials.
59      * @throws OperationNotSupportedException if the user is not logged in.
60      */
61     public void deployCatalog() throws OperationNotSupportedException {
62         if (credentials == null)
63             throw new OperationNotSupportedException("You must be logged in to deploy a new
64                 ↪ catalog.");
65         catalogManager = new CatalogManager(credentials);
66     }
67
68     /**
69      * Returns the user address.
70      * @return a String with the user address.
71      * @throws OperationNotSupportedException if the user is not logged in.
72      */
73     public String getUserAddress() throws OperationNotSupportedException {
74         if (credentials == null)
75             throw new OperationNotSupportedException("You must be logged in.");
76         return credentials.getAddress();
77     }
78
79     /**
80      * Returns true if the user that deployed the catalog is the current user, false otherwise.
81      * @return true if the current user is the catalog owner, false otherwise.
82      * @throws OperationNotSupportedException if the user is not connected to a catalog.
83      */
84     public boolean isCatalogOwner() throws OperationNotSupportedException {
85         if (catalogManager == null)
86             throw new OperationNotSupportedException("Not connected to a catalog.");
87         return catalogManager.getOwner().equals(getUserAddress());
88     }
89
90     /**
91      * Returns the catalog manager.
92      * @return the CatalogManager.
93      */
94     public CatalogManager getCatalogManager() {
95         return catalogManager;
96     }
97
98     /**
99      * Returns the user's private key.
100      * @return a String with the user's private key.
101      */
102     public String getPrivateKey() {
103         return privateKey;
104     }

```



```

105  /**
106   * Set the role for the current user.
107   * @param role an int specifying the role.
108   */
109  public void setRole(int role) {
110      this.role = role;
111  }
112
113  /**
114   * Returns the role for the current user.
115   * @return an int specifying the role.
116   */
117  int getRole() {
118      return role;
119  }
120 }

```

Utils.java

```

1  package com.aldodaquino.cobra.gui;
2
3  import com.aldodaquino.cobra.gui.constants.Images;
4
5  import javax.swing.*;
6  import java.awt.*;
7  import java.io.File;
8
9  /**
10   * Utilities for the GUI.
11   * @author Aldo D'Aquino.
12   * @version 1.0.
13   */
14  public class Utils {
15
16      /**
17       * Show a new centered Window with fixed dimensions and not resizable.
18       * @param title of the Window.
19       * @param panel to show in the Window body.
20       * @param exitOnClose if true exit the Client when the Window is close.
21       * @return the generated JFrame.
22       */
23  public static JFrame newWindow(String title, JComponent panel, boolean exitOnClose) {
24      JFrame window = new JFrame(title); // create a window
25      window.setIconImage(Images.logo.getImage()); // set logo as
26      ↪ application icon
27      window.setContentPane(panel); // put a panel
28      ↪ inside the window
29      window.pack(); // resize the window based on content size
30      window.setLocationRelativeTo(null); // center the
31      ↪ window
32      if (exitOnClose)
33          window.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE); // exit program
34          ↪ when window gets closed
35      window.setVisible(true); // show it
36      return window;
37  }
38
39  /**

```

```

36     * Shows a dialog, running on another thread.
37     * @param msg the dialog message.
38     */
39     public static void newMessageDialog(String msg) {
40         newDialog("Info", msg, JOptionPane.INFORMATION_MESSAGE);
41     }
42
43     /**
44     * Shows an error dialog, running on another thread.
45     * @param msg the error message.
46     */
47     public static void newErrorDialog(String msg) {
48         newDialog("Error", msg, JOptionPane.WARNING_MESSAGE);
49     }
50
51     // auxiliary function
52     private static void newDialog(String title, String msg, int type) {
53         Thread t = new Thread(() -> JOptionPane.showMessageDialog(null, msg, title, type));
54         t.start();
55     }
56
57     /**
58     * Shows an error dialog that advice that the program will close, then exit the program.
59     * @param errorMessage the error message.
60     */
61     public static void newExitDialog(String errorMessage) {
62         JOptionPane.showMessageDialog(null, errorMessage, "ERROR! Exiting...",
63             JOptionPane.ERROR_MESSAGE);
64         System.err.println("Exiting from the program. Reason: " + errorMessage);
65         System.exit(1);
66     }
67
68     /**
69     * Shows a confirmation dialog (yes/no).
70     * @param msg the question.
71     * @return true for yes, false for no.
72     */
73     @SuppressWarnings("BooleanMethodIsAlwaysInverted")
74     public static boolean newConfirmDialog(String msg) {
75         return JOptionPane.showConfirmDialog(null, msg, "Warning", JOptionPane.YES_NO_OPTION) ==
76             JOptionPane.YES_OPTION;
77     }
78
79     /**
80     * Set the font size of a label.
81     * @param label of which set the font size.
82     * @param fontSize the size to set.
83     */
84     public static void setFontSize(JLabel label, int fontSize) {
85         label.setFont(new Font(label.getFont().getName(), Font.PLAIN, fontSize));
86     }
87
88     /**
89     * Show the file selection dialog for file choosing and saving.
90     * @return a File.
91     */
92     public static File openFileDialog() {
93         return fileDialog(true, null);
94     }

```

```

95
96  /**
97   * Show the file selection dialog for file choosing and saving.
98   * @param defaultName specify the original filename of the incoming file.
99   * @return a File.
100  */
101  public static File saveFileDialog(String defaultName) {
102      return openFileDialog(false, defaultName);
103  }
104
105  // auxiliary function
106  private static File openFileDialog(boolean isOpenDialog, String filename) {
107      File selected = null;
108      boolean aFileIsSelected = false;
109
110      JFileChooser chooser = null;
111      LookAndFeel previousLF = UIManager.getLookAndFeel();
112      try {
113          UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
114          chooser = new JFileChooser();
115          UIManager.setLookAndFeel(previousLF);
116      } catch (IllegalAccessException | UnsupportedLookAndFeelException |
117      ↪ InstantiationException |
118      ↪ ClassNotFoundException e) {
119          e.printStackTrace();
120      }
121
122      if (chooser == null) chooser = new JFileChooser();
123      if (filename != null) chooser.setSelectedFile(new File(filename));
124
125      do {
126          chooser.setFileSelectionMode(JFileChooser.FILES_ONLY);
127          int returnStatus = isOpenDialog ? chooser.showOpenDialog(null) :
128          ↪ chooser.showSaveDialog(null);
129
130          if (returnStatus == JFileChooser.APPROVE_OPTION)
131              selected = chooser.getSelectedFile();
132          else continue;
133
134          if (!isOpenDialog && selected.exists()) {
135              aFileIsSelected = newConfirmDialog("The file will be overwritten. Are you
136              ↪ sure?");
137          } else if (!isOpenDialog && selected.exists() && !selected.canWrite()) {
138              newErrorDialog("Can't write in the specified path. Please try again.");
139          } else if (isOpenDialog && !selected.canRead()) {
140              newErrorDialog("Can't read the selected file. Please try again.");
141          } else {
142              aFileIsSelected = true;
143          }
144      } while (!aFileIsSelected);
145
146      return selected;
147  }

```

2.4.1 DAPP/gui/components

AsyncPanel.java

```
1 package com.aldodaquino.cobra.gui.components;
2
3 import javax.swing.*;
4 import javax.swing.event.AncestorEvent;
5 import javax.swing.event.AncestorListener;
6 import java.awt.*;
7
8 /**
9  * A JPanel that listen for ancestor changes. If it is added to an ancestor it saves it a
10  ↪ protected variable window.
11  * Include a doAsync method that runs the Runnable in another Thread and set the glass panel of
12  ↪ the window in a loading
13  * state to prevent actions from the user.
14  * @author Aldo D'Aquino.
15  * @version 1.0.
16  */
17 public class AsyncPanel extends JPanel {
18
19     protected JFrame window;
20
21     /**
22     * Constructor.
23     */
24     protected AsyncPanel() {
25
26         addAncestorListener(new AncestorListener() {
27
28             @Override
29             public void ancestorAdded(AncestorEvent event) {
30                 Component ancestor = event.getAncestor();
31                 if (ancestor.getClass() == JFrame.class)
32                     window = (JFrame) ancestor;
33             }
34
35             @Override
36             public void ancestorRemoved(AncestorEvent event) {
37                 window = null;
38             }
39
40             @Override
41             public void ancestorMoved(AncestorEvent event) {
42                 ancestorAdded(event);
43             }
44         });
45     }
46
47     /**
48     * Run a runnable asynchronously. Shows a loading panel during the loading.
49     * @param runnable to be run.
50     */
51     protected void doAsync(Runnable runnable) {
52         startLoading(window);
53         new Thread(() -> {
54             runnable.run();
55             stopLoading(window);
56         })
```

```

55     }).start();
56 }
57
58 private static void startLoading(JFrame window) {
59     if (window == null) return;
60     window.setGlassPane(ComponentFactory.newSpinner());
61     window.getGlassPane().setVisible(true);
62 }
63
64 private static void stopLoading(JFrame window) {
65     if (window == null) return;
66     window.getGlassPane().setVisible(false);
67 }
68 }

```

AuthorContentTable.java

```

1  package com.aldodaquino.cobra.gui.components;
2
3  import com.aldodaquino.cobra.gui.Status;
4  import com.aldodaquino.cobra.gui.panels.AuthorInfoPanel;
5  import com.aldodaquino.cobra.gui.panels.GenreInfoPanel;
6  import com.aldodaquino.cobra.main.Content;
7
8  import javax.swing.*;
9  import javax.swing.table.TableCellRenderer;
10 import java.awt.*;
11 import java.awt.event.MouseAdapter;
12 import java.awt.event.MouseEvent;
13 import java.util.List;
14
15 /**
16  * A JTable for the {@link com.aldodaquino.cobra.gui.panels.AuthorPanel}.
17  * @author Aldo D'Aquino.
18  * @version 1.0.
19  */
20 public class AuthorContentTable extends JTable {
21
22     private static final String[] colNames = {"Address", "Name", "Author", "Genre", "Views",
23 ↵      "Enjoy", "Price fairness",
24         "Content meaning", "Price"};
25
26     private static Object[][] prepareRows(List<Content> contents) {
27         Object[][] rows = new Object[contents.size()][colNames.length];
28         for (int i = 0; i < contents.size(); i++) {
29             String address = contents.get(i).address;
30             rows[i][0] = address;
31             rows[i][1] = contents.get(i).name;
32             rows[i][2] = contents.get(i).author;
33             rows[i][3] = contents.get(i).genre;
34             rows[i][4] = contents.get(i).views;
35             rows[i][5] = contents.get(i).enjoy;
36             rows[i][6] = contents.get(i).priceFairness;
37             rows[i][7] = contents.get(i).contentMeaning;
38             rows[i][8] = contents.get(i).price;
39         }
40         return rows;
41     }
42 }

```

```

41
42 /**
43  * Constructor.
44  * @param status the Status object.
45  * @param contents a List of Content objects.
46  */
47 public AuthorContentTable(Status status, List<Content> contents) {
48     super(prepareRows(contents), colNames);
49
50     // render author and genre as link style
51     TableCellRenderer linkRenderer = (table, value, arg2, arg3, arg4, arg5) ->
52         new JLabel("<html><a href=\"about:\" + value + "\">\" + value + "</a>");
53     getColumnModel().getColumn(2).setCellRenderer(linkRenderer);
54     getColumnModel().getColumn(3).setCellRenderer(linkRenderer);
55
56     // mouse listener for author and genre click and hover
57     addMouseListener(new MouseAdapter() {
58         @Override
59         public void mouseClicked(MouseEvent e) {
60             int row = rowAtPoint(new Point(e.getX(), e.getY()));
61             int col = columnAtPoint(new Point(e.getX(), e.getY()));
62             String cellContent = (String) getModel().getValueAt(row, col);
63             if (col == 2) AuthorInfoPanel.newWindow(status, cellContent);
64             if (col == 3) GenreInfoPanel.newWindow(status, cellContent);
65         }
66
67         @Override
68         public void mouseEntered(MouseEvent e) {
69             int col = columnAtPoint(new Point(e.getX(), e.getY()));
70             if (col == 2 || col == 3) {
71                 setCursor(new Cursor(Cursor.HAND_CURSOR));
72             }
73         }
74
75         @Override
76         public void mouseExited(MouseEvent e) {
77             int col = columnAtPoint(new Point(e.getX(), e.getY()));
78             if (col != 2 && col != 3) {
79                 setCursor(new Cursor(Cursor.DEFAULT_CURSOR));
80             }
81         }
82     });
83 }
84
85 // Make cells not editable
86 @Override
87 public boolean isCellEditable(int row, int column) {
88     return false;
89 }
90
91 }

```

CatalogForm.java

```

1 package com.aldodaquino.cobra.gui.components;
2
3 import com.aldodaquino.cobra.gui.Utills;
4 import com.aldodaquino.cobra.gui.constants.Dimensions;

```

```

5
6 import javax.swing.*;
7 import java.util.function.Consumer;
8
9 /**
10  * A JPanel used in the {@link StarPanel}. Ask the user which catalog to connect to.
11  * @author Aldo D'Aquino.
12  * @version 1.0.
13  */
14 public class CatalogForm extends JPanel {
15
16     private final JTextField catalogAddressField;
17
18     private final Consumer<String> connectCallback;
19
20     /**
21      * Constructor.
22      * @param connectCallback a callback to be invoked when the deploy button is clicked or the
23      * → form is submitted and
24      * the catalog address is correct.
25      * @param deployCallback a callback invoked when the deploy button is clicked.
26      */
27     public CatalogForm(Consumer<String> connectCallback, Runnable deployCallback) {
28         this.connectCallback = connectCallback;
29
30         // set layout (vertical)
31         setLayout(new BorderLayout(this, BorderLayout.Y_AXIS));
32
33         // Labels
34         JLabel catalogAddressLabel = new JLabel("Catalog address:");
35
36         // catalogManager address field: on enter connect
37         catalogAddressField = ComponentFactory.newTextField(e -> connect());
38
39         // Buttons
40         JButton connectButton = ComponentFactory.newButton("Connect", e -> connect());
41         JButton deployButton = ComponentFactory.newButton("Deploy", e -> deployCallback.run());
42
43         // titled border panel for catalogManager connection
44         JPanel connectPanel = ComponentFactory.newTitledBorderPanel("Existent catalogManager");
45         connectPanel.add(ComponentFactory.newVSpacer(Dimensions.V_SPACER_S));
46         connectPanel.add(catalogAddressLabel);
47         connectPanel.add(catalogAddressField);
48         connectPanel.add(ComponentFactory.newVSpacer(Dimensions.V_SPACER_S));
49         connectPanel.add(connectButton);
50
51         // titled border panel for catalogManager connection
52         JPanel deployPanel = ComponentFactory.newTitledBorderPanel("New catalogManager");
53         deployPanel.add(ComponentFactory.newVSpacer(Dimensions.V_SPACER_S));
54         deployPanel.add(deployButton);
55
56         // add all to the panel
57         add(connectPanel);
58         add(ComponentFactory.newVSpacer(Dimensions.V_SPACER_M));
59         add(deployPanel);
60     }
61
62     private void connect() {
63         // get input data

```

```

63     String address = catalogAddressField.getText().trim();
64
65     // add "0x" to the address if not present
66     if (address.length() == 40)
67         address = "0x" + address;
68
69     // check the length of the inputs and validate the form
70     if (address.length() == 42)
71         connectCallback.accept(address);
72     else Utils.newErrorDialog("Check the values entered in the fields.");
73 }
74
75 }

```

ChartWidget.java

```

1  package com.aldodaquino.cobra.gui.components;
2
3  import com.aldodaquino.cobra.gui.Status;
4  import com.aldodaquino.cobra.main.CatalogManager;
5
6  /**
7   * An {@link InfoPanel} used in the {@link com.aldodaquino.cobra.gui.panels.CustomerPanel}.
8   * Shows the chart of the catalog, that includes: the latest content, the most popular content,
9   * ↪ and the highest rated
10  * content in absolute and for each category.
11  * @author Aldo D'Aquino.
12  * @version 1.0.
13  */
14
15  public class ChartWidget extends InfoPanel {
16
17      /**
18       * Constructor.
19       * @param status the Status object.
20       */
21      public ChartWidget(Status status) {
22          super(status, "Charts");
23          CatalogManager catalogManager = status.getCatalogManager();
24
25          latestLabel.update(catalogManager.getLatest());
26          mostPopularLabel.update(catalogManager.getMostPopular());
27          highestRatedLabel.update(catalogManager.getMostRated(null));
28          mostEnjoyedLabel.update(catalogManager.getMostRated("enjoy"));
29          biggestPriceFairnessLabel.update(catalogManager.getMostRated("value for money"));
30          highestContentMeaningLabel.update(catalogManager.getMostRated("content"));
31      }
32  }

```

ComponentFactory.java

```

1  package com.aldodaquino.cobra.gui.components;
2
3  import com.aldodaquino.cobra.gui.constants.Images;
4
5  import javax.swing.*;
6  import javax.swing.border.Border;

```



```

7  import javax.swing.border.TitledBorder;
8  import java.awt.*;
9  import java.awt.event.ActionListener;
10
11  /**
12   * Components factory: makes components creation faster and easier.
13   * @author Aldo D'Aquino.
14   * @version 1.0.
15   */
16  public class ComponentFactory {
17
18      /**
19       * Returns a new JButton.
20       * @param text the text of the button.
21       * @param e the ActionListener of the button.
22       * @return a JButton.
23       */
24      public static JButton newButton(String text, ActionListener e) {
25          JButton button = new JButton(text);
26          button.addActionListener(e);
27          return button;
28      }
29
30      /**
31       * Returns a new JTextField.
32       * @param e the ActionListener called when enter is pressed.
33       * @return a JTextField.
34       */
35      public static JTextField newTextField(ActionListener e) {
36          return newField(e, false);
37      }
38
39      /**
40       * Returns a new JTextField for password. Characters are replaced with dots.
41       * @param e the ActionListener called when enter is pressed.
42       * @return a JTextField.
43       */
44      static JTextField newPasswordField(ActionListener e) {
45          return newField(e, true);
46      }
47
48      // inner class
49      private static JTextField newField(ActionListener e, boolean isPassword) {
50          JTextField field = isPassword ? new JPasswordField() : new JTextField();
51          field.addActionListener(e);
52          return field;
53      }
54
55      /**
56       * Returns a new border with the specified dimensions.
57       * @param width the border width.
58       * @param height the border height.
59       * @return a Border.
60       */
61      public static Border newBorder(int width, int height) {
62          return BorderFactory.createEmptyBorder(height, width, height, width);
63      }
64
65      /**

```

```

66     * Returns a vertical spacer of the specified dimensions.
67     * @param dimension the dimensions.
68     * @return a Component.
69     */
70     public static Component newVSpacer(Dimension dimension) {
71         return Box.createRigidArea(dimension);
72     }
73
74     /**
75     * Returns a panel with a border with the specified title centered and a vertical layout.
76     * @param title the title string.
77     * @return a JPanel.
78     */
79     static JPanel newTitledBorderPanel(String title) {
80         JPanel panel = new JPanel();
81         TitledBorder titledBorder = BorderFactory.createTitledBorder(title);
82         titledBorder.setTitleJustification(TitledBorder.CENTER);
83         panel.setBorder(titledBorder);
84         panel.setLayout(new BoxLayout(panel, BoxLayout.Y_AXIS));
85         return panel;
86     }
87
88     /**
89     * Return a new panel with a centered JLabel containing a loading message with a spinner.
90     * Used in the {@link AsyncPanel}.
91     * @return a JPanel.
92     */
93     static JPanel newSpinner() {
94         JPanel panel = new JPanel();
95         panel.setLayout(new BorderLayout());
96         panel.add(new JLabel("loading... ", Images.loading, JLabel.CENTER));
97         return panel;
98     }
99 }

```

ContentList.java

```

1  package com.aldodaquino.cobra.gui.components;
2
3  import com.aldodaquino.cobra.gui.Status;
4  import com.aldodaquino.cobra.gui.panels.ContentInfoPanel;
5  import com.aldodaquino.cobra.main.Content;
6
7  import javax.swing.*;
8  import java.awt.event.MouseAdapter;
9  import java.awt.event.MouseEvent;
10 import java.util.List;
11
12 /**
13  * a JList of Content objects.
14  * @author Aldo D'Aquino.
15  * @version 1.0.
16  */
17 public class ContentList extends JList<String> {
18
19     private static String[] prepareRows(List<Content> contents) {
20         String[] rows = new String[contents.size()];
21         for (int i = 0; i < contents.size(); i++)

```

```

22         rows[i] = contents.get(i).name;
23     return rows;
24 }
25
26 /**
27  * Constructor.
28  * @param status the Status object.
29  * @param contents a List of Content objects.
30  */
31 public ContentList(Status status, List<Content> contents) {
32     super(prepareRows(contents));
33
34     // double-click listener
35     addMouseListener(new MouseAdapter() {
36         public void mouseClicked(MouseEvent e) {
37             if (e.getClickCount() < 2) return;
38             int index = locationToIndex(e.getPoint());
39             ContentInfoPanel.newWindow(status, contents.get(index).address);
40         }
41     });
42 }
43
44 }

```

InfoPanel.java

```

1  package com.aldodaquino.cobra.gui.components;
2
3  import com.aldodaquino.cobra.gui.Status;
4  import com.aldodaquino.cobra.gui.constants.Dimensions;
5  import com.aldodaquino.cobra.gui.Utills;
6
7  import javax.swing.*;
8  import java.awt.*;
9
10 import static com.aldodaquino.cobra.gui.constants.Dimensions.INFO_PANEL_PADDING;
11
12 /**
13  * Info panel, superclass of {@link com.aldodaquino.cobra.gui.panels.AuthorInfoPanel},
14  * {@link com.aldodaquino.cobra.gui.panels.GenreInfoPanel} and {@link ChartWidget}.
15  * @author Aldo D'Aquino.
16  * @version 1.0.
17  */
18 public class InfoPanel extends JPanel {
19
20     protected final LabelPanel latestLabel;
21     protected final LabelPanel mostPopularLabel;
22     protected final LabelPanel highestRatedLabel;
23     protected final LabelPanel mostEnjoyedLabel;
24     protected final LabelPanel biggestPriceFairnessLabel;
25     protected final LabelPanel highestContentMeaningLabel;
26
27     /**
28      * Constructor. Can be invoked only by its children.
29      * @param status the Status object.
30      * @param mainLabelString the String to be putted in the top of the panel, with a bigger
31      ↪ font.
32      */

```

```

32     protected InfoPanel(Status status, String mainLabelString) {
33
34         // set layout and border
35         setLayout(new GridBagLayout());
36         setBorder(ComponentFactory.newBorder(INFO_PANEL_PADDING.width,
37         → INFO_PANEL_PADDING.height));
38
39         // prepare content
40         JLabel mainLabel = new JLabel(mainLabelString);
41         Utils.setFontSize(mainLabel, mainLabel.getFont().getSize() * 2);
42         latestLabel = new LabelPanel(status, "Latest release: ");
43         mostPopularLabel = new LabelPanel(status, "Most popular content: ");
44         highestRatedLabel = new LabelPanel(status, "Highest rated content: ");
45         mostEnjoyedLabel = new LabelPanel(status, "Most enjoyed content: ");
46         biggestPriceFairnessLabel = new LabelPanel(status, "Biggest value for money content:
47         → ");
48         highestContentMeaningLabel = new LabelPanel(status, "Highest rated for content meaning:
49         → ");
50         JButton registerForNewContentsButton = ComponentFactory.newButton("Register/unregister
51         → for new contents",
52         e -> status.getCatalogManager().listenNewContentAvailable(mainLabelString,
53         → (address, name) ->
54         → Utils.showMessageDialog("New content available: " + name + "."));
55
56         // add all to the panel
57         add(mainLabel, UpgradablePanel.newGBC(1, 1));
58         add(ComponentFactory.newVSpacer(Dimensions.V_SPACER_M), UpgradablePanel.newGBC(1, 2));
59         add(latestLabel, UpgradablePanel.newGBC(1, 3));
60         add(mostPopularLabel, UpgradablePanel.newGBC(1, 4));
61         add(ComponentFactory.newVSpacer(Dimensions.V_SPACER_M), UpgradablePanel.newGBC(1, 5));
62         add(highestRatedLabel, UpgradablePanel.newGBC(1, 6));
63         add(ComponentFactory.newVSpacer(Dimensions.V_SPACER_S), UpgradablePanel.newGBC(1, 7));
64         add(mostEnjoyedLabel, UpgradablePanel.newGBC(1, 8));
65         add(biggestPriceFairnessLabel, UpgradablePanel.newGBC(1, 9));
66         add(highestContentMeaningLabel, UpgradablePanel.newGBC(1, 10));
67         add(registerForNewContentsButton, UpgradablePanel.newGBC(1, 11));
68     }
69 }

```

LabelPanel.java

```

1  package com.aldodaquino.cobra.gui.components;
2
3  import com.aldodaquino.cobra.gui.Status;
4  import com.aldodaquino.cobra.gui.constants.Images;
5  import com.aldodaquino.cobra.gui.panels.ContentInfoPanel;
6  import com.aldodaquino.cobra.main.Content;
7
8  import javax.swing.*;
9  import java.awt.*;
10 import java.awt.event.MouseAdapter;
11 import java.awt.event.MouseEvent;
12
13 /**
14  * Label JPanel, contains a fixed label and an updatable value. The initial value is a loader
15  → spinner.
16  * @author Aldo D'Aquino.

```

```

16  * @version 1.0.
17  */
18  public class LabelPanel extends UpgradablePanel {
19
20      private final JLabel loader = new JLabel(new ImageIcon(Images.loading.getImage()),
21          ↳ JLabel.CENTER);
22      private final GridBagConstraints replacingPosition = newGBC(2, 1);
23
24      private final Status status;
25
26      /**
27       * Constructor.
28       * @param status the Status object.
29       * @param label the label for the value.
30       */
31      LabelPanel(Status status, String label) {
32          this.status = status;
33          add(new JLabel(label), newGBC(1, 1));
34          add(loader, replacingPosition);
35      }
36
37      /**
38       * Set the content name as value of the panel in a link style.
39       * @param content the Content object.
40       */
41      public void update(Content content) {
42          JLabel link = new JLabel(content == null ? ""
43              : "<html><a href=\"about:\" + content.address + \">\" + content.name + "</a>");
44          // onClick show content panel
45          if (content != null) link.addMouseListener(new MouseAdapter() {
46              @Override
47              public void mouseClicked(MouseEvent e) {
48                  ContentInfoPanel.newWindow(status, content.address);
49              }
50              @Override
51              public void mouseEntered(MouseEvent e) {
52                  setCursor(new Cursor(Cursor.HAND_CURSOR));
53              }
54              @Override
55              public void mouseExited(MouseEvent e) {
56                  setCursor(new Cursor(Cursor.DEFAULT_CURSOR));
57              }
58          });
59          replaceComponent(loader, link, replacingPosition);
60      }
61  }
62

```

LoginForm.java

```

1  package com.aldodaquino.cobra.gui.components;
2
3  import com.aldodaquino.cobra.gui.Utills;
4  import com.aldodaquino.cobra.gui.constants.Dimensions;
5
6  import javax.swing.*;
7  import java.util.function.Consumer;

```

```

8
9  /**
10   * Login Form under the Logo in the Starter Panel.
11   * @author Aldo D'Aquino.
12   * @version 1.0.
13   */
14  public class LoginForm extends JPanel {
15
16      private final JTextField privateKeyInput;
17      private final Consumer<String> loginCallback;
18
19      /**
20       * Constructor.
21       * @param loginCallback a String Consumer called if the login button is clicked or enter is
22       * pressed and the private
23       * key is valid.
24       */
25      public LoginForm(Consumer<String> loginCallback) {
26          this.loginCallback = loginCallback;
27
28          // set layout (vertical)
29          setLayout(new BoxLayout(this, BoxLayout.Y_AXIS));
30
31          // label over the input fields
32          JLabel privateKeyLabel = new JLabel("Private key:");
33
34          // private key field: on enter login
35          privateKeyInput = ComponentFactory.newPasswordField(e -> login());
36
37          // send button
38          JButton sendButton = ComponentFactory.newButton("Login", e -> login());
39
40          // add all to the panel
41          add(privateKeyLabel);
42          add(privateKeyInput);
43          add(ComponentFactory.newVSpacer(Dimensions.V_SPACER_S));
44          add(sendButton);
45      }
46
47      /**
48       * Submit form action.
49       * Called when the login button is clicked or enter key is pressed from the private key
50       * field.
51       */
52      private void login() {
53          // get input data
54          String privateKey = privateKeyInput.getText().trim();
55
56          // check the length of the inputs and validate the form
57          if (privateKey.length() == 64) loginCallback.accept(privateKey);
58          else Utils.newErrorDialog("Check the values entered in the fields.");
59      }
60  }

```

Logo.java

```

1  package com.aldodaquino.cobra.gui.components;
2

```

```

3  import com.aldodaquino.cobra.gui.Utills;
4  import com.aldodaquino.cobra.gui.constants.Dimensions;
5  import com.aldodaquino.cobra.gui.constants.Images;
6  import com.aldodaquino.cobra.gui.constants.Strings;
7
8  import javax.swing.*;
9  import java.awt.*;
10
11  /**
12   * Application Logo for the {@link com.aldodaquino.cobra.gui.panels.StarterPanel}.
13   * @author Aldo D'Aquino.
14   * @version 1.0.
15   */
16  public class Logo extends JPanel {
17
18      /**
19       * Constructor.
20       */
21      public Logo() {
22          // icon
23          Image icon = Images.logo.getImage()
24              .getScaledInstance(Dimensions.LOGO_SIZE, Dimensions.LOGO_SIZE,
25                  ↪ Image.SCALE_SMOOTH);
26          JLabel iconLabel = new JLabel(new ImageIcon(icon), JLabel.CENTER);
27
28          // title
29          JLabel title = new JLabel(Strings.appName);
30          title.setForeground(Color.BLACK);
31          title.setHorizontalAlignment(JLabel.CENTER);
32          // find out how much the font can grow in width and calculate the corresponding font
33          ↪ size
34          Font labelFont = title.getFont();
35          String labelText = title.getText();
36          int stringWidth = title.getFontMetrics(labelFont).stringWidth(labelText);
37          double widthRatio = (double) Dimensions.LOGO_SIZE / (double) stringWidth;
38          int newFontSize = (int) (labelFont.getSize() * widthRatio);
39          // set the new font size
40          Utils.setFontSize(title, newFontSize);
41
42          // put components in a container
43          JPanel container = new JPanel();
44          container.setLayout(new BoxLayout(container, BoxLayout.Y_AXIS));
45          container.add(iconLabel);
46          container.add(title);
47
48          // prepare this panel
49          setAlignmentX(Component.CENTER_ALIGNMENT);
50          add(container);
51      }
52  }

```

NewContentsWidget.java

```

1  package com.aldodaquino.cobra.gui.components;
2
3  import com.aldodaquino.cobra.gui.Status;
4  import com.aldodaquino.cobra.gui.Utills;
5  import com.aldodaquino.cobra.main.CatalogManager;

```

```

6  import com.aldodaquino.cobra.main.Content;
7
8  import javax.swing.*;
9  import java.util.List;
10
11  /**
12   * A widget inserted in the {@link com.aldodaquino.cobra.gui.panels.CustomerPanel} under the
13   * → {@link ChartWidget}.
14   * Allows to select how many content the user want in the new content list and shows a window
15   * → with this content.
16   * @author Aldo D'Aquino.
17   * @version 1.0.
18   */
19  public class NewContentsWidget extends JPanel {
20
21      private final Status status;
22      private final CatalogManager catalogManager;
23      private final JSpinner numberSpinner;
24
25      /**
26       * Constructor.
27       * @param status the Status object.
28       */
29      public NewContentsWidget(Status status) {
30          this.status = status;
31          catalogManager = status.getCatalogManager();
32
33          JLabel label1 = new JLabel("Get");
34          SpinnerNumberModel spinnerModel = new SpinnerNumberModel(10, 0, 100, 1);
35          numberSpinner = new JSpinner(spinnerModel);
36          JLabel label2 = new JLabel("new contents");
37          JButton goButton = ComponentFactory.newButton("Go", e -> getNewContentList());
38
39          add(label1);
40          add(numberSpinner);
41          add(label2);
42          add(goButton);
43      }
44
45      private void getNewContentList() {
46          List<Content> contents = catalogManager.getNewContentList((int)
47          → numberSpinner.getValue());
48          JList contentList = new ContentList(status, contents);
49          Utils.newWindow("New content list", contentList, false);
50      }
51  }

```

RoleForm.java

```

1  package com.aldodaquino.cobra.gui.components;
2
3  import com.aldodaquino.cobra.gui.constants.Dimensions;
4
5  import javax.swing.*;
6
7  /**
8   * The last of the Panels shows in the {@link com.aldodaquino.cobra.gui.panels.StarterPanel}.

```



```

9      * Asks the user if want to see the {@link com.aldodaquino.cobra.gui.panels.CustomerPanel} or
↪    the
10     * {@link com.aldodaquino.cobra.gui.panels.AuthorPanel}.
11     * @author Aldo D'Aquino.
12     * @version 1.0.
13     */
14     public class RoleForm extends JPanel {
15
16         /**
17          * Constructor.
18          * @param browseCallback callback for the "Browse contents" button.
19          * @param manageCallback callback for the "Manage my contents" button.
20          * @param disconnectCallback callback for the "Disconnect" button.
21          * @param deleteCallback callback for the "Delete catalogManager" contents button. If null
↪    the button is not shown.
22          */
23         public RoleForm(Runnable browseCallback, Runnable manageCallback, Runnable
↪    disconnectCallback,
24                         Runnable deleteCallback) {
25
26             // set layout (vertical)
27             setLayout(new BoxLayout(this, BoxLayout.Y_AXIS));
28
29             // Labels
30             JLabel title = new JLabel("What do you want to do?");
31
32             // Buttons
33             JButton browseButton = ComponentFactory.newButton("Browse contents", e ->
↪    browseCallback.run());
34             JButton manageButton = ComponentFactory.newButton("Manage my contents", e ->
↪    manageCallback.run());
35             JButton disconnectButton = ComponentFactory.newButton("Disconnect from catalogManager",
36                         e -> disconnectCallback.run());
37
38             // add all to the panel
39             add(title);
40             add(ComponentFactory.newVSpacer(Dimensions.V_SPACER_S));
41             add(browseButton);
42             add(ComponentFactory.newVSpacer(Dimensions.V_SPACER_S));
43             add(manageButton);
44             add(ComponentFactory.newVSpacer(Dimensions.V_SPACER_M));
45             add(disconnectButton);
46
47             // only for catalogManager owner
48             if (deleteCallback != null) {
49                 add(ComponentFactory.newVSpacer(Dimensions.V_SPACER_S));
50                 add(ComponentFactory.newVSpacer(Dimensions.V_SPACER_L));
51                 JButton deleteButton = ComponentFactory.newButton("Delete catalogManager", e ->
↪    deleteCallback.run());
52                 add(deleteButton);
53             }
54         }
55     }
56 }

```

StarPanel.java

```

1 package com.aldodaquino.cobra.gui.components;
2

```

```

3  import com.aldodaquino.cobra.gui.constants.Dimensions;
4  import com.aldodaquino.cobra.gui.constants.Images;
5
6  import java.awt.event.MouseAdapter;
7  import java.awt.event.MouseEvent;
8  import java.util.ArrayList;
9  import java.util.Collections;
10 import java.util.List;
11
12 import javax.swing.*;
13
14 /**
15  * Shows rating or ask the user to vote.
16  * @author Aldo D'Aquino.
17  * @version 1.0.
18  */
19 public class StarPanel extends UpgradablePanel {
20
21     private static final int STARS_NUMBER = 5;
22
23     private int rating;
24     private final boolean enabled;
25     private final List<JLabel> stars = new ArrayList<>(Collections.nCopies(5, null));
26
27     /**
28      * Constructor. Initialize an empty StarPanel, enabled.
29      */
30     public StarPanel() {
31         this(0, true);
32     }
33
34     /**
35      * Constructor. Initialize a StarPanel with the specified rating, disabled.
36      * @param rating in the interval [0, 5]. 0 means no rating.
37      */
38     public StarPanel(int rating) {
39         this(rating, false);
40     }
41
42     /**
43      * Constructor.
44      * @param rating the rating in the interval [0, 5]. 0 means no rating.
45      * @param enabled true if users can vote, false to show only the rating.
46      */
47     private StarPanel(int rating, boolean enabled) {
48         stars.add(null);
49
50         setRating(rating);
51         this.enabled = enabled;
52     }
53
54     /**
55      * Returns the rating.
56      * @return int in the interval [0, 5]. 0 means no rating.
57      */
58     public int getRating() {
59         return rating;
60     }
61

```

```

62     private void setRating(int rating) {
63         if (rating < 0 || rating > STARS_NUMBER)
64             throw new IllegalArgumentException("The rating must appertain at the interval [0, "
65                 + STARS_NUMBER + "].");
66         this.rating = rating;
67         for (int i = 0; i < STARS_NUMBER; i++) {
68             JLabel newStar = getStar(i < rating);
69             replaceComponent(stars.get(i), newStar, newGBC(i + 1, 0));
70             stars.add(i, newStar);
71         }
72     }
73     private JLabel getStar(boolean filled) {
74         ImageIcon icon = filled ? Images.filledStar : Images.emptyStar;
75         icon = Images.getScaled(icon, Dimensions.STAR_SIZE);
76         JLabel label = new JLabel(icon, JLabel.CENTER);
77         if (enabled) label.addMouseListener(new MouseAdapter() {
78             @Override
79             public void mouseClicked(MouseEvent e) {
80                 setRating(stars.indexOf(label) + 1);
81             }
82         });
83         return label;
84     }
85 }
86

```

UpgradablePanel.java

```

1  package com.aldodaquino.cobra.gui.components;
2
3  import java.awt.*;
4
5  /**
6   * An {@link AsyncPanel} with {@link GridBagLayout}. Implements method to easy replace panels.
7   * @author Aldo D'Aquino.
8   * @version 1.0.
9   */
10 public class UpgradablePanel extends AsyncPanel {
11
12     /**
13      * Constructor. Set the GridBagLayout. Can be called only by its children.
14      */
15     protected UpgradablePanel() {
16         setLayout(new GridBagLayout());
17     }
18
19     /**
20      * Replace a component with another one. Can be called only by its children.
21      * @param toBeReplaced the old component.
22      * @param replacement the new component.
23      * @param position the position of the component to be replaced.
24      */
25     protected void replaceComponent(Component toBeReplaced, Component replacement,
26         ↪ GridBagConstraints position) {
27         if (replacement == null) return;
28         if (toBeReplaced != null) remove(toBeReplaced);
29         add(replacement, position);
30     }
31

```

```

29         if (window != null) {
30             window.revalidate();
31             window.repaint();
32             window.pack();
33         }
34     }
35
36     /**
37      * A public method that help to create GridBagConstraints in less time.
38      * @param x the gridx property of the GridBagConstraints object.
39      * @param y the gridy property of the GridBagConstraints object.
40      * @return a GridBagConstraints object.
41      */
42     public static GridBagConstraints newGBC(int x, int y) {
43         GridBagConstraints gbc = new GridBagConstraints();
44         gbc.gridx = x;
45         gbc.gridy = y;
46         return gbc;
47     }
48 }

```

UserInfo.java

```

1  package com.aldodaquino.cobra.gui.components;
2
3  import com.aldodaquino.cobra.gui.constants.Colors;
4  import com.aldodaquino.cobra.gui.Status;
5  import com.aldodaquino.cobra.gui.constants.Dimensions;
6
7  import javax.naming.OperationNotSupportedException;
8  import javax.swing.*;
9  import java.awt.*;
10
11  /**
12   * Shows the catalog to which the user is connected and the user address and premium status.
13   * @author Aldo D'Aquino.
14   * @version 1.0.
15   */
16  public class UserInfo extends UpgradablePanel {
17
18      private final Status status;
19
20      private JLabel catalogAddressLabel;
21      private final GridBagConstraints catalogAddressPosition;
22
23      private JLabel accountAddressLabel;
24      private final GridBagConstraints accountAddressPosition;
25
26      private JLabel premiumLabel;
27      private final GridBagConstraints premiumPosition;
28
29      /**
30       * Constructor.
31       * @param status the Status object.
32       */
33      public UserInfo(Status status) {
34          this.status = status;
35      }

```

```

36 // catalog label
37 JLabel catalog = new JLabel("Catalog:");
38 catalogAddressLabel = newCatalogAddressLabel();
39 catalogAddressPosition = UpgradablePanel.newGBC(1, 2);
40
41 // account label
42 JLabel accountLabel = new JLabel("Account:");
43 accountAddressLabel = newAccountAddressLabel();
44 accountAddressPosition = UpgradablePanel.newGBC(1, 5);
45
46 // premium label
47 premiumLabel = newPremiumLabel();
48 premiumPosition = UpgradablePanel.newGBC(1, 6);
49
50 // add to the panel
51 add(catalog, UpgradablePanel.newGBC(1, 1));
52 add(catalogAddressLabel, catalogAddressPosition);
53 add(ComponentFactory.newVSpacer(Dimensions.V_SPACER_S), UpgradablePanel.newGBC(1, 3));
54 add(accountLabel, UpgradablePanel.newGBC(1, 4));
55 add(accountAddressLabel, accountAddressPosition);
56 add(premiumLabel, premiumPosition);
57 }
58
59 /**
60  * Update its fields according with the latest information available in status.
61  */
62 public void updateStatus() {
63     JLabel newCatalogAddressLabel = newCatalogAddressLabel();
64     replaceComponent(catalogAddressLabel, newCatalogAddressLabel, catalogAddressPosition);
65     catalogAddressLabel = newCatalogAddressLabel;
66
67     JLabel newAccountAddressLabel = newAccountAddressLabel();
68     replaceComponent(accountAddressLabel, newAccountAddressLabel, accountAddressPosition);
69     accountAddressLabel = newAccountAddressLabel;
70
71     JLabel newPremiumLabel = newPremiumLabel();
72     replaceComponent(premiumLabel, newPremiumLabel, premiumPosition);
73     premiumLabel = newPremiumLabel;
74 }
75
76 private JLabel newAccountAddressLabel() {
77     String account;
78     try {
79         account = status.getUserAddress();
80     } catch (OperationNotSupportedException e) {
81         // not logged in
82         account = "not logged in";
83     }
84     return new JLabel(account);
85 }
86
87 private JLabel newCatalogAddressLabel() {
88     return new JLabel(status.getCatalogManager() == null ? "not connected"
89         : status.getCatalogManager().getAddress());
90 }
91
92 private JLabel newPremiumLabel() {
93     if (status.getCatalogManager() == null) return new JLabel(); // catalog not
94     ↪ connected

```

```

94     Boolean isPremium = status.getCatalogManager().isPremium();
95     JLabel newPremiumLabel;
96     if (isPremium) {
97         newPremiumLabel = new JLabel("Premium user");
98         newPremiumLabel.setForeground(Colors.GREEN);
99     } else {
100         newPremiumLabel = new JLabel("Not Premium user");
101         newPremiumLabel.setForeground(Colors.RED);
102     }
103     return newPremiumLabel;
104 }
105
106 }

```

ViewsContentTable.java

```

1  package com.aldodaquino.cobra.gui.components;
2
3  import com.aldodaquino.cobra.gui.Status;
4  import com.aldodaquino.cobra.gui.panels.ContentInfoPanel;
5  import com.aldodaquino.cobra.main.Content;
6
7  import javax.swing.*;
8  import java.awt.*;
9  import java.awt.event.MouseAdapter;
10 import java.awt.event.MouseEvent;
11 import java.util.List;
12
13 /**
14  * A content list that includes also the number of views.
15  * Accessible in the {@link com.aldodaquino.cobra.gui.panels.CustomerPanel}.
16  * @author Aldo D'Aquino.
17  * @version 1.0.
18  */
19 public class ViewsContentTable extends JTable {
20
21     private static final String[] colNames = {"Name", "Views"};
22
23     private static Object[][] prepareRows(List<Content> contents) {
24         Object[][] rows = new Object[contents.size()][colNames.length];
25         for (int i = 0; i < contents.size(); i++) {
26             rows[i][0] = contents.get(i).name;
27             rows[i][1] = contents.get(i).views;
28         }
29         return rows;
30     }
31
32     /**
33      * Constructor.
34      * @param status the Status object.
35      * @param contents a List of Content objects to be shown in this table.
36      */
37     public ViewsContentTable(Status status, List<Content> contents) {
38         super(prepareRows(contents), colNames);
39
40         // double-click listener
41         addMouseListener(new MouseAdapter() {
42             @Override

```

```

43         public void mouseClicked(MouseEvent e) {
44             if (e.getClickCount() < 2) return;
45             int row = rowAtPoint(new Point(e.getX(), e.getY()));
46             ContentInfoPanel.newWindow(status, contents.get(row).address);
47         }
48     });
49 }
50
51 // Make cells not editable
52 @Override
53 public boolean isCellEditable(int row, int column) {
54     return false;
55 }
56
57 }

```

2.4.2 DAPP/gui/constants

Colors.java

```
1 package com.aldodaquino.cobra.gui.constants;
2
3 import java.awt.*;
4
5 /**
6  * Colors constants for the Graphic Interface.
7  * @author Aldo D'Aquino.
8  * @version 1.0.
9  */
10 public class Colors {
11
12     public static final Color GREEN = new Color(0, 150, 0);
13     public static final Color RED = new Color(255, 40, 40);
14
15 }
```

Dimensions.java

```
1 package com.aldodaquino.cobra.gui.constants;
2
3 import java.awt.*;
4
5 /**
6  * Dimensions constant for the Graphic Interface.
7  * @author Aldo D'Aquino.
8  * @version 1.0.
9  */
10 public class Dimensions {
11
12     // icons size
13     public static final int LOGO_SIZE = 128;
14     public static final int STAR_SIZE = 10;
15
16     // borders padding
17     public static final Dimension STARTER_PANEL_PADDING = new Dimension(60, 30);
18     public static final Dimension INFO_PANEL_PADDING = new Dimension(30, 30);
19     public static final Dimension LATERAL_BAR_PADDING = new Dimension(15, 15);
20
21     // spacers and separators
22     public static final Dimension V_SPACER_S = new Dimension(0,5);
23     public static final Dimension V_SPACER_M = new Dimension(0,15);
24     public static final Dimension V_SPACER_L = new Dimension(0,35);
25
26 }
```

Images.java

```
1 package com.aldodaquino.cobra.gui.constants;
2
3
4 import javax.swing.*;
5 import java.awt.*;
6
```



```

7  /**
8   * Images used in the Graphic Interface.
9   * @author Aldo D'Aquino.
10  * @version 1.0.
11  */
12  public class Images {
13
14      public static final ImageIcon logo = new ImageIcon(Images.class.getResource("/logo.png"));
15      public static final ImageIcon loading = new
16      ↪ ImageIcon(Images.class.getResource("/loading.gif"));
17      public static final ImageIcon emptyStar = new
18      ↪ ImageIcon(Images.class.getResource("/empty-star.png"));
19      public static final ImageIcon filledStar = new
20      ↪ ImageIcon(Images.class.getResource("/filled-star.png"));
21
22      /**
23       * Returns the scaled version of an image.
24       * @param icon the original image.
25       * @param size the size that you want the final image to have.
26       * @return another ImageIcon, scaled.
27       */
28      public static ImageIcon getScaled(ImageIcon icon, int size) {
29          return new ImageIcon(icon.getImage().getScaledInstance(size, size,
30          ↪ Image.SCALE_SMOOTH));
31      }
32  }

```

Strings.java

```

1  package com.aldodaquino.cobra.gui.constants;
2
3  /**
4   * Strings constants for the app.
5   * @author Aldo D'Aquino.
6   * @version 1.0.
7   */
8  public class Strings {
9
10     public static final String appName = "COBrA DAPP";
11
12 }

```

2.4.3 DAPP/gui/panels

AuthorInfoPanel.java

```
1 package com.aldodaquino.cobra.gui.panels;
2
3 import com.aldodaquino.cobra.gui.Status;
4 import com.aldodaquino.cobra.gui.Utills;
5 import com.aldodaquino.cobra.gui.components.InfoPanel;
6 import com.aldodaquino.cobra.main.CatalogManager;
7
8 /**
9  * Shows the author charts in the catalog.
10  * @see InfoPanel the parent class.
11  * @author Aldo D'Aquino.
12  * @version 1.0.
13  */
14 public class AuthorInfoPanel extends InfoPanel {
15
16     static final String WINDOW_TITLE = "About the author";
17
18     /**
19      * Constructor.
20      * @param status the Status object.
21      * @param author the author address.
22      */
23     AuthorInfoPanel(Status status, String author) {
24         super(status, author);
25         CatalogManager catalogManager = status.getCatalogManager();
26
27         new Thread(() -> latestLabel.update(catalogManager.getLatestByAuthor(author))).start();
28         new Thread(() ->
29             ↳ mostPopularLabel.update(catalogManager.getMostPopularByAuthor(author)).start();
30             ↳ highestRatedLabel.update(catalogManager.getMostRatedByAuthor(author,
31             ↳ null))).start();
32             ↳ mostEnjoyedLabel.update(catalogManager.getMostRatedByAuthor(author,
33             ↳ "enjoy"))).start();
34             ↳ biggestPriceFairnessLabel.update(catalogManager.getMostRatedByAuthor(author,
35             ↳ "value for money"))).start();
36             ↳ highestContentMeaningLabel.update(catalogManager.getMostRatedByAuthor(author,
37             ↳ "content"))).start();
38     }
39
40     /**
41      * Open a new window with this panel.
42      * @param status the Status.
43      * @param author of the content.
44      */
45     public static void newWindow(Status status, String author) {
46         Utills.newWindow(WINDOW_TITLE, new AuthorInfoPanel(status, author), false);
47     }
48 }
```

AuthorPanel.java

```
1 package com.aldodaquino.cobra.gui.panels;
2
```

```

3 import com.aldodaquino.cobra.gui.components.AsyncPanel;
4 import com.aldodaquino.cobra.gui.components.AuthorContentTable;
5 import com.aldodaquino.cobra.gui.components.ComponentFactory;
6 import com.aldodaquino.cobra.gui.Utills;
7 import com.aldodaquino.cobra.main.CatalogManager;
8 import com.aldodaquino.cobra.main.Content;
9 import com.aldodaquino.cobra.gui.Status;
10 import com.aldodaquino.cobra.main.ContentManager;
11
12 import java.math.BigInteger;
13 import java.util.List;
14 import javax.naming.OperationNotSupportedException;
15 import javax.swing.*;
16
17 /**
18  * The author panel, a main panel showed after the starter panel if the user have chosen the
19  * ↪ author role.
20  * @author Aldo D'Aquino.
21  * @version 1.0.
22  */
23 public class AuthorPanel extends AsyncPanel {
24
25     private final Status status;
26     private final CatalogManager catalogManager;
27
28     private final JScrollPane tableContainer;
29     private JTable table;
30
31     /**
32      * Constructor.
33      * @param status the Status object.
34      */
35     public AuthorPanel(Status status) {
36         this.status = status;
37         catalogManager = status.getCatalogManager();
38
39         // get the content list
40         List<Content> contents;
41         try {
42             contents = catalogManager.getAuthorContents(status.getUserAddress());
43         } catch (OperationNotSupportedException e) {
44             throw new RuntimeException(e);
45         }
46
47         // listen events
48         catalogManager.listenCatalogClosed(() -> Utills.newExitDialog("Catalog closed."));
49         for (Content content : contents)
50             catalogManager.listenPaymentAvailable(content.address,
51                 (addr, name) -> Utills.newMessageDialog("Payment available for content " +
52                     ↪ name + "."));
53
54         // table container
55         table = new AuthorContentTable(status, contents);
56         tableContainer = new JScrollPane();
57         tableContainer.setViewportViewView(table);
58
59         // buttons
60         JPanel buttonsPad = new JPanel();
61         buttonsPad.setLayout(new BoxLayout(buttonsPad, BoxLayout.X_AXIS));

```

```

60     JButton deployButton = ComponentFactory.newButton("Deploy a new content", e ->
        ↪ deployContent());
61     JButton updateButton = ComponentFactory.newButton("Update table", e -> updateTable());
62     JButton withdrawButton = ComponentFactory.newButton("Withdraw selected", e ->
        ↪ withdrawSelected());
63     buttonsPad.add(deployButton);
64     buttonsPad.add(updateButton);
65     buttonsPad.add(withdrawButton);
66
67     // assemble the panel
68     setLayout(new BoxLayout(this, BoxLayout.Y_AXIS));
69     add(tableContainer);
70     add(buttonsPad);
71 }
72
73 private void deployContent() {
74     JPanel deployContentPanel = new DeployContentPanel(status, this::onDeployed);
75     Utils.newWindow("Deploy new content", deployContentPanel, false);
76 }
77
78 private void onDeployed(String address) {
79     // when deployed listen for payment available on this content
80     ContentManager contentManager = new ContentManager(status.credentials, address);
81     contentManager.listenContentPublished(() -> {
82         catalogManager.listenPaymentAvailable(address, (addr, name) ->
83             Utils.showMessageDialog("Payment available for content " + name + "."));
84         updateTable();
85     });
86 }
87
88 private void updateTable() {
89     doAsync(() -> {
90         try {
91             List<Content> contents =
92                 ↪ catalogManager.getAuthorContents(status.getUserAddress());
93             table = new AuthorContentTable(status, contents);
94             tableContainer.setViewportView(table);
95         } catch (OperationNotSupportedException e) {
96             e.printStackTrace();
97             Utils.newErrorDialog(e.getMessage());
98             System.exit(-1);
99         }
100     });
101 }
102
103 private void withdrawSelected() {
104     doAsync(() -> {
105         String address = table.getValueAt(table.getSelectedRow(), 0).toString();
106         BigInteger amount = catalogManager.withdraw(address);
107         if (amount.equals(BigInteger.ZERO))
108             Utils.newErrorDialog("There is no payout available for this contract.");
109         else Utils.showMessageDialog(amount + " wei collected.");
110     });
111 }
112 }

```

```

1 package com.aldodaquino.cobra.gui.panels;
2
3 import com.aldodaquino.cobra.connections.API;
4 import com.aldodaquino.cobra.connections.CobraHttpHelper;
5 import com.aldodaquino.cobra.gui.Status;
6 import com.aldodaquino.cobra.gui.Utills;
7 import com.aldodaquino.cobra.gui.components.AsyncPanel;
8 import com.aldodaquino.cobra.gui.components.ComponentFactory;
9 import com.aldodaquino.cobra.gui.components.StarPanel;
10 import com.aldodaquino.cobra.gui.components.UpgradablePanel;
11 import com.aldodaquino.cobra.gui.constants.Dimensions;
12 import com.aldodaquino.cobra.main.CatalogManager;
13 import com.aldodaquino.cobra.main.Content;
14 import com.aldodaquino.cobra.main.ContentManager;
15 import com.aldodaquino.javautils.FileExchange;
16
17 import javax.swing.*;
18 import java.awt.*;
19 import java.awt.event.MouseAdapter;
20 import java.awt.event.MouseEvent;
21 import java.io.File;
22 import java.util.HashMap;
23 import java.util.Map;
24
25 import static com.aldodaquino.cobra.gui.components.UpgradablePanel.newGBC;
26 import static com.aldodaquino.cobra.gui.constants.Dimensions.INFO_PANEL_PADDING;
27
28 /**
29  * Shows info about a content. Allows also to buy and consume the content or gift it to another
30  * → user.
31  * @author Aldo D'Aquino.
32  * @version 1.0.
33  */
34 public class ContentInfoPanel extends AsyncPanel {
35
36     private static final String WINDOW_TITLE = "About the author";
37
38     private final Status status;
39     private final CatalogManager catalogManager;
40     private final Content content;
41     private final ContentManager contentManager;
42
43     /**
44      * Constructor.
45      * @param status the Status object.
46      * @param address the content address.
47      */
48     private ContentInfoPanel(Status status, String address) {
49         this.status = status;
50         catalogManager = status.getCatalogManager();
51         content = catalogManager.getContentInfo(address);
52         contentManager = new ContentManager(status.credentials, address);
53
54         // prepare content
55         JLabel mainLabel = new JLabel(content.name);
56         Utills.setFontSize(mainLabel, mainLabel.getFont().getSize() * 2);
57         JLabel addressLabel = new JLabel("Address: " + content.address);
58         JPanel authorLabel = prepareLink("Author: ", content.author,
59             new AuthorInfoPanel(status, content.author, AuthorInfoPanel.WINDOW_TITLE));

```

```

59     JPanel genreLabel = prepareLink("Genre: ", content.genre,
60         new GenreInfoPanel(status, content.genre), GenreInfoPanel.WINDOW_TITLE);
61     JLabel priceLabel = new JLabel("price: " + content.price);
62     JLabel viewsLabel = new JLabel("Views: " + content.views);
63     JPanel averageRatingLabel = prepareStar("Average rating: ", content.averageRating);
64     JPanel enjoyLabel = prepareStar("Enjoy: ", content.enjoy);
65     JPanel priceFairnessLabel = prepareStar("Value for money: ", content.priceFairness);
66     JPanel contentMeaningLabel = prepareStar("Content meaning: ", content.contentMeaning);
67     JButton viewButton = ComponentFactory.newButton("View", e -> view());
68     JButton giftButton = ComponentFactory.newButton("Gift this content", e -> gift());
69
70     // prepare the panel
71     setLayout(new GridBagLayout());
72     setBorder(ComponentFactory.newBorder(INFO_PANEL_PADDING.width,
73         ↳ INFO_PANEL_PADDING.height));
74     add(mainLabel, newGBC(1, 1));
75     add(ComponentFactory.newVSpacer(Dimensions.V_SPACER_M), newGBC(1, 2));
76     add(ComponentFactory.newVSpacer(Dimensions.V_SPACER_M), newGBC(1, 4));
77     add(addressLabel, newGBC(1, 5));
78     add(authorLabel, newGBC(1, 6));
79     add(genreLabel, newGBC(1, 7));
80     add(priceLabel, newGBC(1, 8));
81     add(ComponentFactory.newVSpacer(Dimensions.V_SPACER_M), newGBC(1, 9));
82     add(viewsLabel, newGBC(1, 10));
83     add(ComponentFactory.newVSpacer(Dimensions.V_SPACER_M), newGBC(1, 11));
84     add(averageRatingLabel, newGBC(1, 12));
85     add(enjoyLabel, newGBC(1, 13));
86     add(priceFairnessLabel, newGBC(1, 14));
87     add(contentMeaningLabel, newGBC(1, 15));
88     add(ComponentFactory.newVSpacer(Dimensions.V_SPACER_M), newGBC(1, 16));
89     add(viewButton, newGBC(1, 17));
90     add(giftButton, newGBC(1, 18));
91
92 }
93
94 /**
95  * Open a new window with this panel.
96  * @param status the Status.
97  * @param address of the content.
98  */
99 public static void newWindow(Status status, String address) {
100     Utils.newWindow(WINDOW_TITLE, new ContentInfoPanel(status, address), false);
101 }
102
103 private void view() {
104     doAsync(() -> {
105         // Check if the user has access and ask for buy if he hasn't
106         if (!catalogManager.isPremium() && !catalogManager.hasAccess(content.address)) {
107             if (!Utils.newConfirmDialog("You don't have access to this content. Do you want
108                 ↳ to buy it for "
109                 + content.price + "?")) return; // doesn't have access and doesn't want
110                 ↳ to buy the access
111             if (catalogManager.buyContent(content.address, content.price))
112                 catalogManager.listenAccessGranted((addr, name) -> {
113                     Utils.showMessageDialog("Content bought.");
114                     retrieveContent();
115                 });
116             else Utils.newErrorDialog("Cannot buy this content. You may have bought it
117                 ↳ previously.");
118         }
119     });
120 }

```

```

114         } else retrieveContent();
115     });
116 }
117
118 private void retrieveContent() {
119     // make the request
120     Map<String, String> parameters = new HashMap<>();
121     parameters.put("privateKey", status.getPrivateKey());
122     parameters.put("address", content.address);
123
124     String hostname = contentManager.getHostname();
125     int port = contentManager.getPort();
126     if (port == 0) {
127         Utils.newErrorDialog("The content has an invalid port number. Cannot contact the
128             ↪ author's server.");
129         return;
130     }
131     String url = "http://" + hostname + ":" + port + API.ACCESS_API_PATH;
132
133     // get the response and retrieve the socket port number
134     CobraHttpHelper.Response response = CobraHttpHelper.makePost(url, parameters);
135     if (response.code != 200) Utils.newErrorDialog("HTTP ERROR " + response.code + ": " +
136         ↪ response.data);
137     Map<String, String> map = CobraHttpHelper.parseJson(response.data);
138     int socketPort = Integer.parseInt(map.get("port"));
139     String filename = map.get("filename");
140
141     // download the file
142     File file = Utils.saveFileDialog(filename);
143     FileExchange.receiveFile(file, hostname, socketPort);
144 }
145
146 private void gift() {
147     JPanel pickUserPanel = new PickUserPanel((String user) ->
148         doAsync(() -> {
149             if (catalogManager.giftContent(content.address, user, content.price))
150                 catalogManager.listenAccessGranted(user, (address, name) ->
151                     Utils.newMessageDialog("Content " + name + " gifted to " + user
152                         ↪ + "."));
153             else Utils.newErrorDialog("Cannot gift this content. The user may have
154                 ↪ already bought it.");
155         }));
156     Utils.newWindow("Gift content", pickUserPanel, false);
157 }
158
159 private JPanel prepareLink(String label, String value, JPanel onClickPanel, String
160     ↪ windowTitle) {
161     JLabel link = new JLabel("<html><a href=\"about:\" + value + "\">\" + value + "</a>");
162     link.addMouseListener(new MouseAdapter() {
163         @Override
164         public void mouseClicked(MouseEvent e) {
165             Utils.newWindow(windowTitle, onClickPanel, false);
166         }
167         @Override
168         public void mouseEntered(MouseEvent e) {
169             setCursor(new Cursor(Cursor.HAND_CURSOR));
170         }
171     });
172     @Override

```

```

168         public void mouseExited(MouseEvent e) {
169             setCursor(new Cursor(Cursor.DEFAULT_CURSOR));
170         }
171     });
172     JPanel panel = new JPanel(new GridBagLayout());
173     panel.add(new JLabel(label), UpgradablePanel.newGBC(1, 1));
174     panel.add(link, UpgradablePanel.newGBC(2, 1));
175     return panel;
176 }
177
178 private JPanel prepareStar(String label, int rating) {
179     JPanel panel = new JPanel(new GridBagLayout());
180     panel.add(new JLabel(label), UpgradablePanel.newGBC(1, 1));
181     panel.add(new StarPanel(rating), UpgradablePanel.newGBC(2, 1));
182     return panel;
183 }
184
185 }

```

CustomerPanel.java

```

1 package com.aldodaquino.cobra.gui.panels;
2
3 import com.aldodaquino.cobra.gui.components.*;
4 import com.aldodaquino.cobra.gui.constants.Dimensions;
5 import com.aldodaquino.cobra.gui.Utills;
6 import com.aldodaquino.cobra.main.CatalogManager;
7 import com.aldodaquino.cobra.gui.Status;
8
9 import javax.swing.*;
10 import java.awt.*;
11
12 import static com.aldodaquino.cobra.gui.constants.Dimensions.LATERAL_BAR_PADDING;
13
14 /**
15  * The customer panel, a main panel showed after the starter panel if the user have chosen the
16  * ↪ customer role.
17  * @author Aldo D'Aquino.
18  * @version 1.0.
19  */
20 public class CustomerPanel extends UpgradablePanel {
21
22     private final Status status;
23     private final CatalogManager catalogManager;
24
25     private final JScrollPane tableContainer;
26     private Component table;
27     private final JPanel lateralBar;
28     private final UserInfo userInfo;
29     private ChartWidget chartWidget;
30     private final GridBagConstraints chartWidgetPosition;
31
32     private boolean showViews = false;
33
34     /**
35      * Constructor.
36      * @param status the Status object.
37      */

```



```

37 public CustomerPanel(Status status) {
38     this.status = status;
39     catalogManager = status.getCatalogManager();
40
41     // listen for events
42     catalogManager.listenCatalogClosed(() -> Utils.newExitDialog("Catalog closed.));
43     catalogManager.listenNewContentAvailable((name, address) -> update());
44     catalogManager.listenFeedbackAvailable((address, name) ->
45         Utils.newWindow("Vote content", new VotingPanel(address, name, catalogManager),
46             ↪ false));
47
48     // table container
49     tableContainer = new JScrollPane();
50     table = getTable();
51     tableContainer.setViewportViewView(table);
52
53     // lateral bar
54     userInfo = new UserInfo(status);
55     JButton buyPremiumButton = ComponentFactory.newButton("Buy premium", e ->
56         ↪ buyPremium());
57     JButton giftPremiumButton = ComponentFactory.newButton("Gift premium", e ->
58         ↪ giftPremium());
59     JButton updateButton = ComponentFactory.newButton("Refresh", e -> update());
60     JButton showHideViewsButton = ComponentFactory.newButton("Show/hide views", e -> {
61         showViews = !showViews;
62         update();
63     });
64     chartWidget = new ChartWidget(status);
65     JPanel newContentWidget = new NewContentsWidget(status);
66
67     lateralBar = new JPanel(new GridBagLayout()) {
68         // prevent widely resize with window
69         @Override
70         public Dimension getMaximumSize() {
71             Dimension dim = super.getMaximumSize();
72             dim.width = getPreferredSize().width;
73             return dim;
74         }
75         // minimum size to fit all component
76         @Override
77         public Dimension getMinimumSize() {
78             return getPreferredSize();
79         }
80     };
81
82     lateralBar.add(userInfo, newGBC(1, 1));
83     lateralBar.setBorder(ComponentFactory.newBorder(LATERAL_BAR_PADDING.width,
84         ↪ LATERAL_BAR_PADDING.height));
85     lateralBar.add(ComponentFactory.newVSpacer(Dimensions.V_SPACER_M), newGBC(1, 2));
86     lateralBar.add(buyPremiumButton, newGBC(1, 3));
87     lateralBar.add(giftPremiumButton, newGBC(1, 4));
88     lateralBar.add(ComponentFactory.newVSpacer(Dimensions.V_SPACER_L), newGBC(1, 5));
89     lateralBar.add(updateButton, newGBC(1, 6));
90     lateralBar.add(showHideViewsButton, newGBC(1, 7));
91     lateralBar.add(ComponentFactory.newVSpacer(Dimensions.V_SPACER_L), newGBC(1, 8));
92     chartWidgetPosition = newGBC(1, 9);
93     lateralBar.add(chartWidget, chartWidgetPosition);
94     lateralBar.add(newContentWidget, newGBC(1, 10));

```



```

4 import com.aldodaquino.cobra.gui.Status;
5 import com.aldodaquino.cobra.gui.components.AsyncPanel;
6 import com.aldodaquino.cobra.gui.components.ComponentFactory;
7 import com.aldodaquino.cobra.gui.constants.Dimensions;
8 import com.aldodaquino.cobra.gui.Utills;
9 import com.aldodaquino.cobra.connections.CobraHttpHelper;
10 import com.aldodaquino.javautils.FileExchange;
11
12 import javax.swing.*;
13 import java.io.File;
14 import java.math.BigInteger;
15 import java.nio.channels.ServerSocketChannel;
16 import java.util.HashMap;
17 import java.util.Map;
18 import java.util.function.Consumer;
19
20 /**
21  * Panel to deploy a new content.
22  * @author Aldo D'Aquino.
23  * @version 1.0.
24  */
25 class DeployContentPanel extends AsyncPanel {
26
27     // fields
28     private final JTextField addressField;
29     private final JTextField portField;
30     private final JTextField nameField;
31     private final JTextField genreField;
32     private final JTextField priceField;
33
34     private final Status status;
35     private final Consumer<String> deployCallback;
36
37     private File file;
38
39     /**
40      * Constructor.
41      * @param status the Status object.
42      * @param deployCallback a Consumer of content address, called after that the content has
43      * ↪ been deployed.
44      */
45     DeployContentPanel(Status status, Consumer<String> deployCallback) {
46         this.status = status;
47         this.deployCallback = deployCallback;
48
49         // set layout (vertical)
50         setLayout(new BoxLayout(this, BoxLayout.Y_AXIS));
51
52         // label over the input fields
53         JLabel introLabel = new JLabel("The deployed content will be placed in your author
54         ↪ server. Specify the " +
55         ↪ "address and the port (default 8080) of a server running an author server
56         ↪ instance. The author " +
57         ↪ "server must always be online so that the content is accessible.");
58         JLabel addressLabel = new JLabel("Address (IP or domain:");
59         JLabel portLabel = new JLabel("Port:");
60         JLabel nameLabel = new JLabel("Name:");
61         JLabel genreLabel = new JLabel("Genre:");
62         JLabel priceLabel = new JLabel("Price:");

```

```

60     JLabel selectFileLabel = new JLabel("Pick the content file:");
61
62     // input field
63     priceField = ComponentFactory.newTextField(e -> {});
64     genreField = ComponentFactory.newTextField(e -> priceField.grabFocus());
65     nameField = ComponentFactory.newTextField(e -> genreField.grabFocus());
66     portField = ComponentFactory.newTextField(e -> nameField.grabFocus());
67     portField.setText("8080");
68     addressField = ComponentFactory.newTextField(e -> portField.grabFocus());
69     addressField.setText("localhost");
70
71     // buttons
72     JButton selectFileButton = ComponentFactory.newButton("Open", e -> file =
73     ↪     Utils.openFileDialog());
74     JButton sendButton = ComponentFactory.newButton("Deploy", e -> deploy());
75
76     // add all to the panel
77     add(introLabel);
78     add(ComponentFactory.newVSpacer(Dimensions.V_SPACER_L));
79     add(addressLabel);
80     add(addressField);
81     add(ComponentFactory.newVSpacer(Dimensions.V_SPACER_S));
82     add(portLabel);
83     add(portField);
84     add(ComponentFactory.newVSpacer(Dimensions.V_SPACER_S));
85     add(nameLabel);
86     add(nameField);
87     add(ComponentFactory.newVSpacer(Dimensions.V_SPACER_S));
88     add(genreLabel);
89     add(genreField);
90     add(ComponentFactory.newVSpacer(Dimensions.V_SPACER_S));
91     add(priceLabel);
92     add(priceField);
93     add(ComponentFactory.newVSpacer(Dimensions.V_SPACER_S));
94     add(selectFileLabel);
95     add(selectFileButton);
96     add(ComponentFactory.newVSpacer(Dimensions.V_SPACER_S));
97     add(sendButton);
98
99     /**
100     * Submit form action.
101     * Called when the deploy button is clicked.
102     */
103     private void deploy() {
104         // get input data
105         String url = addressField.getText().trim();
106         if (url.length() == 0) {
107             Utils.newErrorDialog("You must specify an url.");
108             return;
109         }
110         try {
111             String portS = portField.getText().trim();
112             int port = portS.equals("") ? 8080 : Integer.parseInt(portS);
113             if (port <= 0) throw new NumberFormatException();
114             url = "http://" + url + ":" + port + API.DEPLOY_API_PATH;
115         } catch (NumberFormatException e) {
116             Utils.newErrorDialog("Invalid port number.");
117             return;

```

```

118     }
119
120     String name = nameField.getText().trim();
121     if (name.length() == 0) {
122         Utils.newErrorDialog("You must specify a name.");
123         return;
124     }
125     String genre = genreField.getText().trim();
126     String priceS = priceField.getText().trim();
127     BigInteger price;
128     try {
129         price = new BigInteger(priceS.length() != 0 ? priceS : "0");
130     } catch (NumberFormatException e) {
131         Utils.newErrorDialog("Invalid port number.");
132         return;
133     }
134
135     // open the socket for the file
136     if (file == null) {
137         Utils.newErrorDialog("You must choose a file to be uploaded.");
138         return;
139     }
140
141     ServerSocketChannel serverSocketChannel = FileExchange.openFileSocket();
142     if (serverSocketChannel == null) {
143         Utils.newErrorDialog("Error while opening server socket.");
144         return;
145     }
146
147     int port = serverSocketChannel.socket().getLocalPort();
148     FileExchange.startFileSender(serverSocketChannel, file,
149         () -> Utils.newMessageDialog("File uploaded successfully.));
150
151     // make the request
152     Map<String, String> parameters = new HashMap<>();
153     parameters.put("privateKey", status.getPrivateKey());
154     parameters.put("name", name);
155     parameters.put("genre", genre);
156     parameters.put("price", price.toString());
157     parameters.put("port", Integer.toString(port));
158     parameters.put("filename", file.getName());
159
160     CobraHttpHelper.Response response = CobraHttpHelper.makePost(url, parameters);
161     if (response.code != 200) {
162         Utils.newErrorDialog("HTTP ERROR " + response.code + ": " + response.data);
163         return;
164     }
165
166     // close the widow
167     deployCallback.accept(response.data);
168     window.dispose();
169 }
170 }

```

GenreInfoPanel.java

```

1 package com.aldodaquino.cobra.gui.panels;
2

```

```

3 import com.aldodaquino.cobra.gui.Status;
4 import com.aldodaquino.cobra.gui.Utills;
5 import com.aldodaquino.cobra.gui.components.InfoPanel;
6 import com.aldodaquino.cobra.main.CatalogManager;
7
8 /**
9  * Shows the genre charts in the catalog.
10  * @see InfoPanel the parent class.
11  * @author Aldo D'Aquino.
12  * @version 1.0.
13  */
14 public class GenreInfoPanel extends InfoPanel {
15
16     static final String WINDOW_TITLE = "About the author";
17
18     /**
19      * Constructor.
20      * @param status the Status object.
21      * @param genre the genre.
22      */
23     GenreInfoPanel(Status status, String genre) {
24         super(status, genre);
25         CatalogManager catalogManager = status.getCatalogManager();
26
27         new Thread(() -> latestLabel.update(catalogManager.getLatestByGenre(genre))).start();
28         new Thread(() ->
29             ↳ mostPopularLabel.update(catalogManager.getMostPopularByGenre(genre))).start();
30         new Thread(() -> highestRatedLabel.update(catalogManager.getMostRatedByGenre(genre,
31             ↳ null))).start();
32         new Thread(() -> mostEnjoyedLabel.update(catalogManager.getMostRatedByGenre(genre,
33             ↳ "enjoy"))).start();
34         new Thread(() ->
35             ↳ biggestPriceFairnessLabel.update(catalogManager.getMostRatedByGenre(genre,
36                 ↳ "value for money"))).start();
37         new Thread(() ->
38             ↳ highestContentMeaningLabel.update(catalogManager.getMostRatedByGenre(genre,
39                 ↳ "content"))).start();
40     }
41
42     /**
43      * Open a new window with this panel.
44      * @param status the Status.
45      * @param genre of the content.
46      */
47     public static void newWindow(Status status, String genre) {
48         Utills.newWindow(WINDOW_TITLE, new GenreInfoPanel(status, genre), false);
49     }
50 }

```

PickUserPanel.java

```

1 package com.aldodaquino.cobra.gui.panels;
2
3 import com.aldodaquino.cobra.gui.components.AsyncPanel;
4 import com.aldodaquino.cobra.gui.components.ComponentFactory;
5 import com.aldodaquino.cobra.gui.Utills;
6 import com.aldodaquino.cobra.gui.constants.Dimensions;

```

```

7
8 import javax.swing.*;
9 import java.util.function.Consumer;
10
11 /**
12  * Asks the user for another user address.
13  * @author Aldo D'Aquino.
14  * @version 1.0.
15  */
16 class PickUserPanel extends AsyncPanel {
17     private final JTextField addressField;
18     // the callback to call if the input data are correct
19     private final Consumer<String> giftCallback;
20
21     /**
22      * Constructor.
23      * @param giftCallback a Consumer of user address, invoked when the button is clicked or
24      * ↪ enter is pressed if the
25      *           address has a valid format.
26      */
27     PickUserPanel(Consumer<String> giftCallback) {
28         this.giftCallback = giftCallback;
29
30         // set layout (vertical)
31         setLayout(new BorderLayout(this, BorderLayout.Y_AXIS));
32
33         // label over the input fields
34         JLabel addressLabel = new JLabel("User address:");
35
36         // input field
37         addressField = ComponentFactory.newTextField(e -> gift());
38
39         // send button
40         JButton giftButton = ComponentFactory.newButton("Gift", e -> gift());
41
42         // add all to the panel
43         add(addressLabel);
44         add(addressField);
45         add(ComponentFactory.newVSpacer(Dimensions.V_SPACER_S));
46         add(giftButton);
47     }
48
49     /**
50      * Submit form action.
51      * Called when the gift button is clicked or enter key is pressed from the address field.
52      */
53     private void gift() {
54         // get input data
55         String address = addressField.getText().trim();
56
57         // add "0x" to the address if not present
58         if (address.length() == 40)
59             address = "0x" + address;
60
61         // check the length of the inputs and validate the form
62         if (address.length() == 42)
63             giftCallback.accept(address);
64         else Utils.newErrorDialog("You must specify an address.");
65     }
66 }

```

```

65         // close the widow
66         window.dispose();
67     }
68 }

```

StarterPanel.java

```

1  package com.aldodaquino.cobra.gui.panels;
2
3  import com.aldodaquino.cobra.gui.components.*;
4  import com.aldodaquino.cobra.gui.constants.Dimensions;
5  import com.aldodaquino.cobra.gui.Utills;
6  import com.aldodaquino.cobra.gui.Status;
7
8  import javax.naming.OperationNotSupportedException;
9  import javax.swing.*;
10 import java.awt.*;
11 import java.util.function.Consumer;
12
13 import static com.aldodaquino.cobra.gui.constants.Dimensions.STARTER_PANEL_PADDING;
14
15 /**
16  * The starter panel. Main panel showed when the app starts.
17  * Manage the user login and require all the data to start the app.
18  * @author Aldo D'Aquino.
19  * @version 1.0.
20  */
21 public class StarterPanel extends UpgradablePanel {
22
23     private final Status status = new Status();
24     private final Consumer<Status> whenDone;
25
26     private final UserInfo userInfo;
27     private final JPanel loginForm;
28     private JPanel catalogForm;
29     private JPanel roleForm;
30
31     private final GridBagConstraints replacingPosition;
32
33     /**
34      * Constructor.
35      * @param whenDone a consumer of Status object called when all the requested fields in the
36      ↪ status object are
37      * completed.
38      */
39     public StarterPanel(Consumer<Status> whenDone) {
40         this.whenDone = whenDone;
41
42         // init components
43         JPanel logo = new Logo();
44         userInfo = new UserInfo(status);
45         loginForm = new LoginForm(this::loginCallback);
46         replacingPosition = new GBC(1, 5);
47
48         // prepare the panel and add components
49         setBorder(ComponentFactory.newBorder(STARTER_PANEL_PADDING.width,
50 ↪ STARTER_PANEL_PADDING.height));
51         add(logo, new GBC(1, 1));

```



```

50         add(ComponentFactory.newVSpacer(Dimensions.V_SPACER_L), newGBC(1, 2));
51         add(userInfo, newGBC(1, 3));
52         add(ComponentFactory.newVSpacer(Dimensions.V_SPACER_L), newGBC(1, 4));
53         add(loginForm, replacingPosition);
54     }
55
56     // minimum size to fit all component
57     @Override
58     public Dimension getMinimumSize() {
59         return getPreferredSize();
60     }
61
62     /* CALLBACKS */
63     private void loginCallback(String privateKey) {
64         doAsync(() -> {
65             try {
66                 // set the status
67                 status.login(privateKey);
68                 // update the user info
69                 userInfo.updateStatus();
70                 // change form
71                 catalogForm = new CatalogForm(this::connectCallback, this::deployCallback);
72                 replaceComponent(loginForm, catalogForm, replacingPosition);
73             } catch (OperationNotSupportedException e) {
74                 e.printStackTrace();
75                 Utils.newErrorDialog(e.getMessage());
76             }
77         });
78     }
79
80     private void connectCallback(String catalogAddress) {
81         doAsync(() -> {
82             try {
83                 status.connectCatalog(catalogAddress);
84                 postConnect();
85             } catch (OperationNotSupportedException e) {
86                 e.printStackTrace();
87                 Utils.newErrorDialog(e.getMessage());
88             }
89         });
90     }
91
92     private void deployCallback() {
93         doAsync(() -> {
94             try {
95                 status.deployCatalog();
96                 postConnect();
97             } catch (OperationNotSupportedException e) {
98                 e.printStackTrace();
99                 Utils.newErrorDialog(e.getMessage());
100             }
101         });
102     }
103
104     private void postConnect() throws OperationNotSupportedException {
105         userInfo.updateStatus();
106         Runnable deleteCallback = status.isCatalogOwner() ? this::deleteCallback : null;
107         roleForm = new RoleForm(this::browseCallback, this::manageCallback,
            ↪ this::disconnectCallback, deleteCallback);

```

```

108         replaceComponent(catalogForm, roleForm, replacingPosition);
109     }
110
111     private void disconnectCallback() {
112         doAsync(() -> {
113             status.disconnectCatalog();
114             userInfo.updateStatus();
115             replaceComponent(roleForm, catalogForm, replacingPosition);
116         });
117     }
118
119     private void deleteCallback() {
120         if (!Utils.newConfirmDialog("Do you really want to delete this catalog?")) return;
121         doAsync(() -> {
122             if (status.getCatalogManager().suicide()) {
123                 Utils.showMessageDialog("Catalog deleted.");
124                 disconnectCallback();
125             } else Utils.newErrorDialog("UNKNOWN ERROR: the catalog is not deleted.");
126         });
127     }
128
129     private void browseCallback() {
130         status.setRole(Status.ROLE_CUSTOMER);
131         whenDone.accept(status);
132     }
133
134     private void manageCallback() {
135         status.setRole(Status.ROLE_AUTHOR);
136         whenDone.accept(status);
137     }
138 }
139

```

VotingPanel.java

```

1  package com.aldodaquino.cobra.gui.panels;
2
3  import com.aldodaquino.cobra.gui.components.AsyncPanel;
4  import com.aldodaquino.cobra.gui.components.ComponentFactory;
5  import com.aldodaquino.cobra.gui.components.StarPanel;
6  import com.aldodaquino.cobra.gui.constants.Dimensions;
7  import com.aldodaquino.cobra.main.CatalogManager;
8
9  import javax.swing.*;
10
11  /**
12   * A panel to ask the user to vote a content consumed recently.
13   * @author Aldo D'Aquino.
14   * @version 1.0.
15   */
16  class VotingPanel extends AsyncPanel {
17
18      /**
19       * Constructor.
20       * @param contentName the content name.
21       * @param contentAddress the content address.
22       * @param catalogManager the CatalogManager loaded or deployed by the user.
23       */

```

```

24 VotingPanel(String contentName, String contentAddress, CatalogManager catalogManager) {
25
26     // prepare components
27     JLabel infoLabel = new JLabel("You can now rate for the content " + contentName + ".");
28     JLabel enjoyLabel = new JLabel("Enjoy:");
29     StarPanel enjoyStars = new StarPanel();
30     JLabel valueForMoneyLabel = new JLabel("Value for money:");
31     StarPanel valueForMoneyStars = new StarPanel();
32     JLabel contentMeaningLabel = new JLabel("Content meaning:");
33     StarPanel contentMeaningStars = new StarPanel();
34     JButton voteButton = ComponentFactory.newButton("Vote", e -> {
35         catalogManager.vote(contentAddress, enjoyStars.getRating(),
36             ↪ valueForMoneyStars.getRating(),
37                 contentMeaningStars.getRating());
38         window.dispose();
39     });
40
41     // prepare the panel
42     setLayout(new BoxLayout(this, BoxLayout.Y_AXIS));
43     add(infoLabel);
44     add(ComponentFactory.newVSpacer(Dimensions.V_SPACER_L));
45     add(enjoyLabel);
46     add(enjoyStars);
47     add(ComponentFactory.newVSpacer(Dimensions.V_SPACER_S));
48     add(valueForMoneyLabel);
49     add(valueForMoneyStars);
50     add(ComponentFactory.newVSpacer(Dimensions.V_SPACER_S));
51     add(contentMeaningLabel);
52     add(contentMeaningStars);
53     add(ComponentFactory.newVSpacer(Dimensions.V_SPACER_M));
54     add(voteButton);
55 }

```

2.5 DAPP/test (not required)

Main.java

```
1  package com.aldodaquino.cobra.test;
2
3  import com.aldodaquino.javautils.CliHelper;
4  import com.aldodaquino.javautils.FileExchange;
5  import com.aldodaquino.cobra.connections.API;
6  import com.aldodaquino.cobra.connections.CobraHttpHelper;
7  import com.aldodaquino.cobra.gui.Utills;
8  import com.aldodaquino.cobra.main.CatalogManager;
9  import org.web3j.crypto.Credentials;
10
11 import java.io.File;
12 import java.io.IOException;
13 import java.net.URI;
14 import java.net.URISyntaxException;
15 import java.nio.channels.ServerSocketChannel;
16 import java.util.HashMap;
17 import java.util.Map;
18 import java.util.concurrent.ThreadLocalRandom;
19
20 /**
21  * Prepares an environment for tests:
22  * - deploys a CatalogContract;
23  * - starts an author-server on the default port 8080;
24  * - deploy some contents;
25  * - start two instances of the GUI, one for the Customer and one for the Author.
26  * You have to login with the rights private key on the GUIs.
27  * @author Aldo D'Aquino.
28  * @version 1.0.
29  */
30 public class Main {
31
32     private static final boolean START_GUI = true;
33
34     private static final int NUMBER_OF_CONTENTS = 15;
35     private static final String[] genres = {"Comedy", "Romance", "Thriller"};
36     private static URI FILENAME;
37
38     // static constructor
39     static {
40         try {
41             FILENAME = Main.class.getResource("/test_file.png").toURI();
42         } catch (URISyntaxException e) {
43             e.printStackTrace();
44         }
45     }
46
47     // Change this value with your private keys.
48     @SuppressWarnings("SpellCheckingInspection")
49     private static final String CATALOG_OWNER_DEFAULT_KEY =
50         "f34cdbfb031497943b0a0248de50567d7ba1dba45309f7955dea5e57d701c060";
51     @SuppressWarnings("SpellCheckingInspection")
52     private static final String[] AUTHOR_DEFAULT_KEYS =
53         {"9b6f8306cb695bb9939682f65a5e7f35679073ec4d4c99774c18361e183e65ba",
54         "3538c75c66e5e3ec35343da5726efde1682d8db51ffcb69e0f6d92e75fa8a37d"};
55
56     /**
```

```

57  * Starts the tests.
58  * @param args a String[], optionally containing 2 options:
59  *           -p --private-key    Catalog owner private key.
60  *           -a --author         Author's private key.
61  *           If args is null or an option is missing the default option will be used.
62  */
63  public static void main(String[] args) {
64
65      // Parse cmd options
66      CliHelper cliHelper = new CliHelper();
67      cliHelper.addOption("h", "help", false, "Print this help message.");
68      cliHelper.addOption("k", "private-key", true, "Catalog owner private key.");
69      cliHelper.addOption("a", "author", true, "Author's private key.");
70      cliHelper.parse(args);
71
72      String catalogOwnerKeyOpt = cliHelper.getValue("private-key");
73      String catalogOwnerKey = catalogOwnerKeyOpt.length() > 0 ? catalogOwnerKeyOpt :
        ↪ CATALOG_OWNER_DEFAULT_KEY;
74
75      String[] authorKeysOpt = cliHelper.getValues("author");
76      String[] authorKeys = authorKeysOpt.length > 0 ? authorKeysOpt : AUTHOR_DEFAULT_KEYS;
77
78
79      // Create credentials for catalog owner's private key
80      Credentials catalogOwnerCredentials = Credentials.create(catalogOwnerKey);
81      System.out.println("Created credentials for catalog owner. Account address: "
82          + catalogOwnerCredentials.getAddress() + ".\n");
83
84
85      // Deploy a new Catalog and retrieve the address
86      CatalogManager catalogManager = new CatalogManager(catalogOwnerCredentials);
87      String catalogAddress = catalogManager.getAddress();
88      System.out.println("Deployed a new CatalogContract. Catalog address: " + catalogAddress
        ↪ + ".\n");
89
90
91      // Start n author servers on ports 8000-(8000+n-1),
92      // bound with the catalog and associated to the author's private key
93      System.out.println("Starting "+ authorKeys.length + " author servers, please wait...");
94      int[] port = {8000}; // "Clickety-click... Barba-trick!": in lambda expressions we can
        ↪ only use final
95                          // (or effectively final) variables.
96      for (String authorKey : authorKeys) {
97          new Thread(() -> {
98              try {
99                  com.aldodaquino.cobra.authorserver.Main.main(
100                      new String[]{"-k", authorKey, "-c", catalogAddress, "-n",
                        ↪ "localhost",
101                          "-p", Integer.toString(port[0]++)});
102              } catch (IOException e) {
103                  e.printStackTrace();
104              }
105          }).start();
106      }
107
108      // Wait 3 seconds for the server to become online.
109      try {
110          Thread.sleep(3000);
111      } catch (InterruptedException e) {

```

```

112         System.err.println("Interrupted during while waiting the author server becomes
        ↪ online.");
113         e.printStackTrace();
114     }
115
116
117     // Deploying contents
118     for (int i = 0; i < NUMBER_OF_CONTENTS; i++) {
119         int authorIndex = rand(authorKeys.length);
120         deploy(8000 + authorIndex, authorKeys[authorIndex], "Content " + (i+1),
121             genres[rand(genres.length)], Integer.toString(rand(5) * 5000));
122     }
123
124
125     // Start the GUI
126     if (START_GUI) {
127         System.out.print("Starting two GUI windows...");
128         Process GUI1 = newGUIProcess();
129         Process GUI2 = newGUIProcess();
130
131         // Wait for the GUIs to end
132         try {
133             assert GUI1 != null && GUI2 != null;
134             GUI1.waitFor();
135             GUI2.waitFor();
136             System.out.println("    GUI 1 exit with value " + GUI1.exitValue()
137                 + ".\n    GUI 2 exit with value " + GUI2.exitValue() + ".\n\n");
138             System.exit(GUI1.exitValue() + GUI2.exitValue()); // 0 if none fails, 1 if one
139                 ↪ fails, 2 if both fail.
140         } catch (InterruptedException e) {
141             e.printStackTrace();
142             System.exit(1);
143         }
144     }
145
146
147     /* Auxiliary functions */
148
149     private static int rand(int lessThan) {
150         return ThreadLocalRandom.current().nextInt(0, lessThan);
151     }
152
153     private static void deploy(int serverPort, String authorKey, String name, String genre,
154         ↪ String price) {
155         // assemble the url
156         String url = "http://localhost:" + serverPort + API.DEPLOY_API_PATH;
157
158         // prepare the file
159         ServerSocketChannel serverSocketChannel = FileExchange.openFileSocket();
160         if (serverSocketChannel == null) {
161             Utils.newErrorDialog("Error while opening server socket.");
162             return;
163         }
164
165         File file = new File(FILENAME);
166         int port = serverSocketChannel.socket().getLocalPort();
167         FileExchange.startFileSender(serverSocketChannel, file,
168             () -> System.out.println("File uploaded successfully."));

```

```

168
169 // make the request
170 Map<String, String> parameters = new HashMap<>();
171 parameters.put("privateKey", authorKey);
172 parameters.put("name", name);
173 parameters.put("genre", genre);
174 parameters.put("price", price);
175 parameters.put("port", Integer.toString(port));
176 parameters.put("filename", file.getName());
177
178 System.out.println("Deploying a content..." +
179     "\n    Url: " + url +
180     "\n    Author key: " + authorKey +
181     "\n    Name: " + name +
182     "\n    Genre: " + genre +
183     "\n    Price: " + price);
184
185 CobraHttpHelper.Response response = CobraHttpHelper.makePost(url, parameters);
186 if (response.code != 200) System.err.println("Something went wrong. Response" +
187     ↳ response.toString());
188 else System.out.println("Deployed successfully.\n");
189
190 }
191
192 private static Process newGUIProcess() {
193     String javaHome = System.getProperty("java.home");
194     String javaBin = javaHome + File.separator + "bin" + File.separator + "java";
195     String classpath = System.getProperty("java.class.path");
196     String className = com.aldodaquino.cobra.gui.Main.class.getCanonicalName();
197     ProcessBuilder builder = new ProcessBuilder(javaBin, "-cp", classpath, className);
198     try {
199         return builder.start();
200     } catch (IOException e) {
201         e.printStackTrace();
202     }
203     return null;
204 }

```

2.6 javautils

CliHelper.java

```
1 package com.aldodaquino.javautils;
2
3 import java.util.ArrayList;
4 import java.util.Comparator;
5 import java.util.stream.Stream;
6
7 /**
8  * Help parsing the args[]. You can add the option you want retrieve and this class will parse
9  * ↪ it automatically.
10  * @author Aldo D'Aquino.
11  * @version 1.0.
12  */
13 public class CliHelper {
14
15     private final ArrayList<CliOption> cliOptions = new ArrayList<>();
16     private final ArrayList<CliFlag> cliFlags = new ArrayList<>();
17     private int maxLongOptLength = 0;
18
19     /**
20      * Add available option for the CLI.
21      * @param shortOpt like "-v".
22      * @param longOpt like "--verbose".
23      * @param hasValue true if the option must have a value (i.e. -o value),
24      * ↪ false if is an option without value (i.e. --help).
25      * @param description the description of the option to be shown in the help message.
26      * @throws IllegalArgumentException if the shortOpt or the longOpt already exists in
27      * ↪ another option.
28      */
29     public void addOption(String shortOpt, String longOpt, boolean hasValue, String
30     ↪ description) {
31         if (!hasValue) addFlag(shortOpt, longOpt, description);
32         // search for an already existent option
33         for (CliOption cliOption : cliOptions) {
34             if (cliOption.isEqual(shortOpt) || cliOption.isEqual(longOpt))
35                 throw new IllegalArgumentException("Option already exist.");
36         }
37         // add to the options
38         cliOptions.add(new CliOption(shortOpt, longOpt, description));
39     }
40
41     // Internal auxiliary method
42     private void addFlag(String shortOpt, String longOpt, String description) {
43         // search for an already existent option
44         for (CliFlag cliFlag : cliFlags) {
45             if (cliFlag.isEqual(shortOpt) || cliFlag.isEqual(longOpt))
46                 throw new IllegalArgumentException("Option already exist.");
47         }
48         // add to the options
49         cliFlags.add(new CliFlag(shortOpt, longOpt, description));
50     }
51
52     /**
53      * Parse the String[] args of the main and saves the option value.
54      * @param args the main's args.
55      */
56     public void parse(String[] args) {
```



```

54     if (args == null) return;
55     for (int i = 0; i < args.length; i++) {
56         if (isNotAnOption(args[i])) System.err.println("Invalid option " + args[i] + ".");
57         if (i + 1 < args.length && isNotAnOption(args[i + 1])) { // args[i+1] is the value
            ↪ of args[i]
58             for (CliOption cliOption : cliOptions)
59                 if (cliOption.parse(args[i], args[i+1])) break;
60             i++; // skip i + 1: is not an option
61         } else { // args[i] has no value, so is a flag
62             for (CliFlag cliFlag : cliFlags)
63                 if (cliFlag.parse(args[i])) break;
64         }
65     }
66 }
67
68 /**
69  * Return the list of all values of an option. For example -o value1 -o value2 will return
    ↪ [value1, value2].
70  * @param opt the option in the short or the long format (will return the same list).
71  * @return a String[] containing all the values, empty array if the option has no value or
    ↪ null if this option
72  * doesn't exist.
73  */
74 public String[] getValues(String opt) {
75     for (CliOption cliOption : cliOptions)
76         if (cliOption.isEqual(opt)) return cliOption.getValues();
77     return null;
78 }
79
80 /**
81  * Return first value of an option. For example -o value1 -o value2 will return value1.
82  * Use it for functions that are intended usable only once.
83  * @param opt the option in the short or the long format (will return the same list).
84  * @return the first value, empty string if the option has no value or null if this option
    ↪ doesn't exist.
85  */
86 public String getValue(String opt) {
87     String[] values = getValues(opt);
88     return values == null ? null : values.length == 0 ? "" : values[0];
89 }
90
91 /**
92  * Return true if the program is launched with the specified option, false otherwise.
93  * @param opt the option in the short or the long format (will return the same list).
94  * @return true if the program is launched with the specified option, false otherwise.
95  */
96 public boolean isPresent(String opt) {
97     for (CliFlag cliFlag : cliFlags)
98         if (cliFlag.isEqual(opt))
99             return cliFlag.isPresent();
100    return false;
101 }
102
103 /**
104  * Return a formatted help message showing the usage. The message has this format:
105  * usage:
106  * -o --longopt    an option description
107  * -h --help      shows help
108  * @return String of the message.

```

```

109  */
110  public String getHelpMessage() {
111      // create a sorted collection with all the objects
112      ArrayList<CliObject> cliObjects = new ArrayList<>();
113      cliObjects.addAll(cliOptions);
114      cliObjects.addAll(cliFlags);
115      cliObjects.sort(Comparator.comparing(o -> o.shortOpt));
116
117      // prepare the help string
118      StringBuilder stringBuilder = new StringBuilder("Usage:\n");
119      String initialString = "";
120      for (CliObject cliObject : cliObjects) {
121          stringBuilder.append(initialString).append(cliObject.shortOpt).append("
122              ↪ ").append(cliObject.longOpt);
123          // append enough spaces to align the descriptions plus a tab (4 spaces) as
124          ↪ separator
125          for (int i = 0; i < maxLongOptLength - cliObject.longOpt.length() + 4; i++)
126              stringBuilder.append(" ");
127          stringBuilder.append(cliObject.description);
128          initialString = "\n"; // from now append a line break before the new line
129      }
130      return stringBuilder.toString();
131  }
132
133  /**
134   * Return a missing option message with this format:
135   * Missing an option: -o -option description of the option.
136   * @param opt the short or long code of the option that you want (i.e. "o");
137   * @return String of the message.
138   */
139  public String getMissingOptionMessage(String opt) {
140      StringBuilder stringBuilder = new StringBuilder();
141      Stream.concat(cliOptions.stream(), cliFlags.stream()).forEachOrdered(cliObject -> {
142          if (cliObject.isEqual(opt))
143              stringBuilder.append("Missing an
144                  ↪ option:\n").append(cliObject.shortOpt).append(" ")
145                  .append(cliObject.longOpt).append("
146                  ↪ ").append(cliObject.description);
147      });
148      return stringBuilder.toString();
149  }
150
151  // Internal auxiliary method
152  private boolean isNotAnOption(String string) {
153      return !string.contains("--") && !string.contains("-");
154  }
155
156  /* Auxiliary classes */
157
158  private class CliObject {
159      final String shortOpt;
160      final String longOpt;
161      String description;
162
163      CliObject(String shortOpt, String longOpt, String description) {
164          // add minuses if not present in the head of the string
165          this.shortOpt = shortOpt.length() >= 1 && shortOpt.substring(0, 1).equals("-") ?
166              ↪ shortOpt : "-" + shortOpt;

```

```

163         this.longOpt = longOpt.length() >= 2 && longOpt.substring(0, 2).equals("--") ?
            ↳ longOpt : "--" + longOpt;
164
165         if (shortOpt.length() > 2) throw new IllegalArgumentException("Short option must be
            ↳ a single letter.");
166         if (longOpt.length() > maxLongOptLength) maxLongOptLength = longOpt.length();
167
168         this.description = description;
169     }
170
171     boolean isEqual(String opt) {
172         if (opt.length() >= 2 && opt.substring(0, 2).equals("--")) return
            ↳ longOpt.equals(opt); // is a long option
173         if (opt.length() >= 1 && opt.substring(0, 1).equals("-")) return
            ↳ shortOpt.equals(opt); // is a short option
174         return shortOpt.equals("-" + opt) || longOpt.equals("--" + opt); // does not
            ↳ contains minuses
175     }
176 }
177
178 private class CliOption extends CliObject {
179
180     final ArrayList<String> values = new ArrayList<>();
181
182     CliOption(String shortOpt, String longOpt, String description) {
183         super(shortOpt, longOpt, description);
184     }
185
186     boolean parse(String opt, String value) {
187         if (isEqual(opt)) {
188             values.add(value);
189             return true;
190         }
191         return false;
192     }
193
194     String[] getValues() {
195         return values.toArray(new String[0]);
196     }
197 }
198
199 private class CliFlag extends CliObject {
200
201     boolean found;
202
203     CliFlag(String shortOpt, String longOpt, String description) {
204         super(shortOpt, longOpt, description);
205     }
206
207     boolean parse(String opt) {
208         if (isEqual(opt)) {
209             found = true;
210             return true;
211         }
212         return false;
213     }
214
215     boolean isPresent() {
216

```

```

217         return found;
218     }
219
220 }
221
222 }

```

FileExchange.java

```

1  package com.aldodaquino.javautils;
2
3  import java.io.*;
4  import java.net.*;
5  import java.nio.ByteBuffer;
6  import java.nio.channels.FileChannel;
7  import java.nio.channels.ServerSocketChannel;
8  import java.nio.channels.SocketChannel;
9  import java.nio.file.StandardOpenOption;
10
11  /**
12   * Utility for exchanging file.
13   * Contains method to receive data from a Socket and save it to a File and to read a file and
14   * → write data to the socket.
15   * @author Aldo D'Aquino.
16   * @version 1.2.
17   */
18  public class FileExchange {
19
20      /**
21       * Receive and save a file.
22       * @param destFile the destination file where save data.
23       * @param hostname of the sender.
24       * @param port of the sender.
25       */
26      public static void receiveFile(File destFile, String hostname, int port) {
27          Thread asyncWriter = new Thread(() -> {
28              int failedCount = 0;
29              boolean stop = false;
30              do {
31                  try (SocketChannel socket = SocketChannel.open(new InetSocketAddress(hostname,
32                      → port))) {
33                      System.out.println("Started download: " + destFile.getAbsolutePath() +
34                      → ".");
35                      writeFile(socket, destFile);
36                      System.out.println("Download finished.");
37                      stop = true;
38                  } catch (IOException e) {
39                      if (failedCount < 3) {
40                          try {
41                              failedCount++;
42                              Thread.sleep(5000);
43                          } catch (InterruptedException intExc) {
44                              intExc.printStackTrace();
45                              break;
46                          }
47                      }
48                  }
49              }
50              else {
51                  stop = true;
52              }
53          });
54      }
55  }

```

```

47         System.err.println("Can't connect to the sender.");
48         e.printStackTrace();
49
50     }
51 }
52 } while (!stop);
53 });
54 asyncWriter.start();
55 }
56
57 /**
58  * Opens a socket on which waits an incoming connection from the file recipient.
59  * @return a ServerSocketChannel.
60  */
61 public static ServerSocketChannel openFileSocket() {
62     try {
63         ServerSocketChannel serverSocketChannel = ServerSocketChannel.open();
64         serverSocketChannel.bind(null);
65         return serverSocketChannel;
66     }
67     catch (IOException e) {
68         System.err.println("Error while opening a socket for sending file.");
69         e.printStackTrace();
70     }
71
72     return null;
73 }
74
75 /**
76  * When the other client has connected to the socket it behaves as a server and sends the
77  * file.
78  * @param serverSocket the ServerSocketChannel opened in openFileSocket.
79  * @param file the File to be sent.
80  * @param callback optional callback to be run when the upload is finished.
81  */
82 public static void startFileSender(ServerSocketChannel serverSocket, File file, Runnable
83     callback) {
84     Thread listener = new Thread(() -> {
85         try {
86             serverSocket.socket().setSoTimeout(60000); // 1 minute
87             SocketChannel socketChannel = serverSocket.accept();
88             System.out.println("Started upload: " + file.getAbsolutePath() + ".");
89             readFile(file, socketChannel);
90             System.out.println("Upload finished.");
91             if (callback != null) callback.run();
92         } catch (IOException e) {
93             System.err.println("Error while accepting connection to send file.");
94             e.printStackTrace();
95         }
96     });
97     listener.start();
98 }
99
100 /**
101  * Read a File using NIO channels. Send the file to a socket.
102  * @param file File object, the file to be read.
103  * @param outChannel Socket where the file will be sent.
104  * @throws IOException if the file not exists or is not readable.
105  */

```

```

104     public static void readFile(File file, SocketChannel outChannel) throws IOException {
105         FileChannel inChannel = FileChannel.open(file.toPath(), StandardOpenOption.READ);
106         long size = inChannel.size();
107         ByteBuffer sizeBuffer = ByteBuffer.allocate(8);
108         sizeBuffer.putLong(size);
109         sizeBuffer.flip();
110         while (sizeBuffer.hasRemaining()) outChannel.write(sizeBuffer);
111
112         long transferred = 0;
113         while (size - transferred > 0)
114             transferred += inChannel.transferTo(transferred, size - transferred, outChannel);
115     }
116
117     /**
118      * Read the specified Socket using NIO, and save the data to File.
119      * @param inChannel Socket from where data will be read.
120      * @param file File object, destination of the data.
121      * @throws IOException if the file not exists or is not writeable.
122      */
123     public static void writeFile(SocketChannel inChannel, File file) throws IOException {
124         FileChannel outChannel = FileChannel.open(file.toPath(),
125             StandardOpenOption.CREATE, StandardOpenOption.TRUNCATE_EXISTING,
126             ↪ StandardOpenOption.WRITE);
127         ByteBuffer sizeBuffer = ByteBuffer.allocate(8);
128         while (sizeBuffer.hasRemaining())
129             inChannel.read(sizeBuffer);
130         sizeBuffer.flip();
131
132         long size = sizeBuffer.getLong();
133         long transferred = 0;
134
135         while (size - transferred > 0)
136             transferred += outChannel.transferFrom(inChannel, transferred, size - transferred);
137     }
138 }

```

HttpHelper.java

```

1  package com.alododaquino.javautils;
2
3  import com.sun.net.httpserver.HttpExchange;
4
5  import java.io.*;
6  import java.net.HttpURLConnection;
7  import java.net.URL;
8  import java.net.URLDecoder;
9  import java.util.HashMap;
10 import java.util.Map;
11
12 /**
13  * Contains method that help to make http request.
14  * Works with JSON body for POST request and query-style GET parameters.
15  * @author Aldo D'Aquino.
16  * @version 1.1.
17  */
18 public class HttpHelper {
19

```

```

20  /* SERVER SIDE */
21
22  /**
23   * Parse a GET request and return a Map with keys equals to parameters name and values
→ equals to the parameters
24   * values.
25   * @param request the HttpExchange request received by the handler.
26   * @return Map where both keys and values are strings containing the parameters.
27   */
28  public static Map<String, String> parseGET(HttpExchange request) {
29      String query = request.getRequestURI().getRawQuery();
30      if (query == null || query.length() == 0) throw new IllegalArgumentException("Invalid
→ query: null.");
31      return parseQuery(query);
32  }
33
34  /**
35   * Parse a query and return a Map with keys equals to parameters name and values equals to
→ the parameters values.
36   * @param query the String representing the query.
37   * @return Map where both keys and values are strings containing the parameters.
38   */
39  public static Map<String, String> parseQuery(String query) {
40      // HashMap to be filled with all parameters in the query
41      Map<String, String> parameters = new HashMap<>();
42
43      // Split the query in pairs key=value
44      String pairs[] = query.split("&");
45      // Split each pair in key and value and put them in the Map
46      for (String pair : pairs) {
47          String param[] = pair.split("=");
48          if (param.length > 0) {
49              try {
50                  String key = URLDecoder.decode(param[0],
→ System.getProperty("file.encoding"));
51                  String value = null;
52                  if (param.length > 1) value = URLDecoder.decode(param[1],
→ System.getProperty("file.encoding"));
53                  parameters.put(key, value);
54              } catch (UnsupportedEncodingException e) {
55                  e.printStackTrace();
56              }
57          }
58      }
59      return parameters;
60  }
61
62  /**
63   * Parse a POST request and return a Map with keys equals to parameters name and values
→ equals to the parameters
64   * values.
65   * @param request the HttpExchange request received by the handler.
66   * @return Map where both keys and values are strings containing the parameters.
67   */
68  public static Map<String, String> parsePOST(HttpExchange request) {
69      String json = null;
70      try {
71          InputStreamReader isr = new InputStreamReader(request.getRequestBody(), "utf-8");
72          json = new BufferedReader(isr).readLine();

```

```

73     } catch (IOException e) {
74         e.printStackTrace();
75     }
76     return parseJson(json);
77 }
78
79 /**
80  * Parse a JSON and return a Map with keys equals to parameters name and values equals to
81  * the parameters values.
82  * @param json the String representing the stringified JSON.
83  * @return Map where both keys and values are strings containing the parameters.
84  */
85 public static Map<String, String> parseJson(String json) {
86     // parse the JSON body.
87     if (json == null || json.length() == 0) throw new IllegalArgumentException("Invalid
88     ↪ query: null.");
89
90     // HashMap to be filled with all parameters in the query
91     Map<String, String> parameters = new HashMap<>();
92
93     // remove parenthesis and quotes
94     json = json.replace("{", "").replace("}", "").replaceAll("\\\"", "");
95
96     // Split the query in pairs key=value
97     String pairs[] = json.split("[,]");
98     // Split each pair in key and value and put them in the Map
99     for (String pair : pairs) {
100         String param[] = pair.split("[:]");
101         if (param.length > 0) {
102             String key = param[0];
103             String value = null;
104             if (param.length > 1) value = param[1];
105             parameters.put(key, value);
106         }
107     }
108     return parameters;
109 }
110
111 /**
112  * Send a response to an HttpExchange request.
113  * @param request the request.
114  * @param response a String containing the response.
115  * @param code the status code of the response.
116  */
117 public static void sendResponse(HttpExchange request, String response, int code) {
118     try {
119         request.sendResponseHeaders(code, response.length());
120         OutputStream os = request.getResponseBody();
121         os.write(response.getBytes());
122         os.close();
123     } catch (IOException e) {
124         e.printStackTrace();
125     }
126 }
127
128 /**
129  * Send a response to an HttpExchange request.
130  * @param request the request.
131  * @param response a String containing the response..

```



```

130     */
131     public static void sendResponse(HttpExchange request, String response) {
132         sendResponse(request, response, 200);
133     }
134
135
136     /* CLIENT SIDE */
137
138     /**
139      * Make a GET request on the specified url.
140      * @param url the url to be called.
141      * @param parameters a map containing all the parameters that you want to be passed when
142      * the url is called.
143      * @return a {@link Response} object.
144      */
145     public static Response makeGet(String url, Map<String, String> parameters) {
146         return makeRequest(url + queryParameters(parameters), "GET", "");
147     }
148
149     /**
150      * Make a POST request on the specified url.
151      * @param url the url to be called.
152      * @param parameters a map containing all the parameters that you want to be passed in the
153      * body.
154      * @return a {@link Response} object.
155      */
156     public static Response makePost(String url, Map<String, String> parameters) {
157         return makeRequest(url, "POST", jsonParameters(parameters));
158     }
159
160     // internal function
161     private static Response makeRequest(String url, String method, String parameters) {
162         HttpURLConnection connection = null;
163         int status = -1;
164         try {
165             //Create connection
166             connection = (HttpURLConnection) new URL(url).openConnection();
167             connection.setDoOutput(true);
168             connection.setRequestMethod(method);
169             connection.setRequestProperty("Content-Type", "application/json");
170             connection.setRequestProperty("Content-Length",
171                 Integer.toString(parameters.getBytes().length));
172             connection.setRequestProperty("Content-Language", "en-US");
173             connection.setConnectTimeout(5000);
174             connection.setReadTimeout(5000);
175
176             //Send request
177             if(!parameters.equals("")) {
178                 DataOutputStream outputStream = new
179                     DataOutputStream(connection.getOutputStream());
180                 outputStream.writeBytes(parameters);
181                 outputStream.close();
182             }
183
184             //Get Response
185             status = connection.getResponseCode();
186             BufferedReader bufferedReader = new BufferedReader(new
187                 InputStreamReader(connection.getInputStream()));
188             StringBuilder responseData = new StringBuilder();

```

```

184         String line;
185         String separator = "";
186         while ((line = bufferedReader.readLine()) != null) {
187             responseData.append(separator).append(line);
188             separator = "\n";
189         }
190         bufferedReader.close();
191
192         return new Response(status, responseData.toString());
193     }
194     catch (IOException e) {
195         e.printStackTrace();
196         return status < 0 ? null : new Response(status, "");
197     }
198     finally {
199         if (connection != null) connection.disconnect();
200     }
201 }
202
203 /**
204  * Return a stringified JSON with the passed parameters.
205  * @param parameters a Map where both keys and values are strings containing the
206 ↪ parameters.
207  * @return the stringified JSON.
208 */
209 public static String jsonifyParameters(Map<String, String> parameters) {
210     if (parameters == null) return "";
211     StringBuilder stringBuilder = new StringBuilder();
212     stringBuilder.append("{");
213     String separator = "";
214     for (Map.Entry<String, String> entry : parameters.entrySet()) {
215         ↪ stringBuilder.append(separator)
216         ↪ .append("\"").append(entry.getKey()).append("\":").append(entry.getValue()).append(
217         ↪ separator = ", ";
218     }
219     stringBuilder.append("}");
220     return stringBuilder.toString();
221 }
222
223 /**
224  * Return a string containing the parameters in the query format, ready to be appended to
225 ↪ an url for a GET request.
226  * @param parameters a Map where both keys and values are strings containing the
227 ↪ parameters.
228  * @return the String in the url format.
229 */
230 public static String querifyParameters(Map<String, String> parameters) {
231     if (parameters == null || parameters.size() == 0) return "";
232     StringBuilder stringBuilder = new StringBuilder();
233     stringBuilder.append("?");
234     String separator = "";
235     for (Map.Entry<String, String> entry : parameters.entrySet()) {
236         stringBuilder.append(separator)
237         ↪ .append(entry.getKey()).append("=").append(entry.getValue());
238         separator = "&";
239     }
240     return stringBuilder.toString();
241 }

```

```

239
240 /**
241  * Response class returned by the makeGet and makePost method.
242  * Contains two field: the response code and the data String of the response.
243  * @author Aldo D'Aquino.
244  * @version 1.0.
245  */
246 public static class Response {
247     public final int code;
248     public final String data;
249     private Response (int code, String data) {
250         this.code = code;
251         this.data = data;
252     }
253
254     /**
255      * Returns this object as a stringified JSON.
256      * @return a String representing the stringified JSON.
257      */
258     @Override
259     public String toString() {
260         return "{\"code\": \"" + code + "\", \"data\": \"" + data + "\"}";
261     }
262 }
263
264 }

```

3 Other

compile.sh

```
1  #!/usr/bin/env bash
2
3  # configs
4  contracts_path="contracts"
5  java_path="DAPP/contracts/src"
6  java_package="com.aldodaquino.cobra.contracts"
7
8  # prepare the output folder (deleting any previous compiled files)
9  output_path=${contracts_path}/out
10 rm -rf ${output_path}
11 mkdir -p ${output_path}
12
13 # for each file in the contracts_path
14 for file in ${contracts_path}/*.sol; do
15     # get the contract name
16     contract=${file//$contracts_path//}
17     contract=${contract//.sol/}
18     echo ${contract}
19
20     # compile files
21     solc ${file} --bin --abi --optimize -o ${output_path}/${contract}
22
23     # generate Java classes
24     base_path=${output_path}/${contract}/${contract}
25     web3j solidity generate ${base_path}.bin ${base_path}.abi -o ${java_path} -p
26     ↪ ${java_package}
27 done
```

install.sh

```
1  #!/usr/bin/env bash
2
3  bash compile.sh
4
5  cd DAPP
6  rm jar/*
7
8  mvn clean
9  mvn install
10
11 cd author-server
12 mvn clean
13 mvn install
14
15 cd ../gui
16 mvn clean
17 mvn install
18
19 cd ../test
20 mvn clean
21 mvn install
```

LICENSE.

GNU AFFERO GENERAL PUBLIC LICENSE

Version 3, 19 November 2007

Copyright (C) 2007 Free Software Foundation, Inc. <<http://fsf.org/>>
Everyone is permitted to copy and distribute verbatim copies
of **this** license document, but changing it is not allowed.

Preamble

The GNU Affero General Public License is a free, copyleft license **for**
software and other kinds of works, specifically designed to ensure
cooperation with the community in the **case** of network server software.

The licenses **for** most software and other practical works are designed
to take away your freedom to share and change the works. By contrast,
our General Public Licenses are intended to guarantee your freedom to
share and change all versions of a program--to make sure it remains free
software **for** all its users.

When we speak of free software, we are referring to freedom, not
price. Our General Public Licenses are designed to make sure that you
have the freedom to distribute copies of free **software** (and charge **for**
them **if** you wish), that you receive source code or can get it **if** you
want it, that you can change the software or use pieces of it in **new**
free programs, and that you know you can **do** these things.

Developers that use our General Public Licenses protect your rights
with two steps: (1) **assert** copyright on the software, and (2) offer
you **this** License which gives you legal permission to copy, distribute
and/or modify the software.

A secondary benefit of defending all users' freedom is that
improvements made in alternate versions of the program, **if** they
receive widespread use, become available **for** other developers to
incorporate. Many developers of free software are heartened and
encouraged by the resulting cooperation. However, in the **case** of
software used on network servers, **this** result may fail to come about.
The GNU General Public License permits making a modified version and
letting the **public** access it on a server without ever releasing its
source code to the **public**.

The GNU Affero General Public License is designed specifically to
ensure that, in such cases, the modified source code becomes available
to the community. It requires the operator of a network server to
provide the source code of the modified version running there to the
users of that server. Therefore, **public** use of a modified version, on
a publicly accessible server, gives the **public** access to the source
code of the modified version.

An older license, called the Affero General Public License and
published by Affero, was designed to accomplish similar goals. This is
a different license, not a version of the Affero GPL, but Affero has
released a **new** version of the Affero GPL which permits relicensing under
this license.

The precise terms and conditions **for** copying, distribution and
modification follow.

TERMS AND CONDITIONS

60
61 0. Definitions.
62
63 "This License" refers to version 3 of the GNU Affero General Public License.
64
65 "Copyright" also means copyright-like laws that apply to other kinds of
66 works, such as semiconductor masks.
67
68 "The Program" refers to any copyrightable work licensed under this
69 License. Each licensee is addressed as "you". "Licensees" and
70 "recipients" may be individuals or organizations.
71
72 To "modify" a work means to copy from or adapt all or part of the work
73 in a fashion requiring copyright permission, other than the making of an
74 exact copy. The resulting work is called a "modified version" of the
75 earlier work or a work "based on" the earlier work.
76
77 A "covered work" means either the unmodified Program or a work based
78 on the Program.
79
80 To "propagate" a work means to do anything with it that, without
81 permission, would make you directly or secondarily liable for
82 infringement under applicable copyright law, except executing it on a
83 computer or modifying a private copy. Propagation includes copying,
84 distribution (with or without modification), making available to the
85 public, and in some countries other activities as well.
86
87 To "convey" a work means any kind of propagation that enables other
88 parties to make or receive copies. Mere interaction with a user through
89 a computer network, with no transfer of a copy, is not conveying.
90
91 An interactive user interface displays "Appropriate Legal Notices"
92 to the extent that it includes a convenient and prominently visible
93 feature that (1) displays an appropriate copyright notice, and (2)
94 tells the user that there is no warranty for the work (except to the
95 extent that warranties are provided), that licensees may convey the
96 work under this License, and how to view a copy of this License. If
97 the interface presents a list of user commands or options, such as a
98 menu, a prominent item in the list meets this criterion.
99
100 1. Source Code.
101
102 The "source code" for a work means the preferred form of the work
103 for making modifications to it. "Object code" means any non-source
104 form of a work.
105
106 A "Standard Interface" means an interface that either is an official
107 standard defined by a recognized standards body, or, in the case of
108 interfaces specified for a particular programming language, one that
109 is widely used among developers working in that language.
110
111 The "System Libraries" of an executable work include anything, other
112 than the work as a whole, that (a) is included in the normal form of
113 packaging a Major Component, but which is not part of that Major
114 Component, and (b) serves only to enable use of the work with that
115 Major Component, or to implement a Standard Interface for which an
116 implementation is available to the public in source code form. A
117 "Major Component", in this context, means a major essential component
118 (kernel, window system, and so on) of the specific operating system

119 (if any) on which the executable work runs, or a compiler used to
120 produce the work, or an object code interpreter used to run it.

121
122 The "Corresponding Source" for a work in object code form means all
123 the source code needed to generate, install, and (for an executable
124 work) run the object code and to modify the work, including scripts to
125 control those activities. However, it does not include the work's
126 System Libraries, or general-purpose tools or generally available free
127 programs which are used unmodified in performing those activities but
128 which are not part of the work. For example, Corresponding Source
129 includes interface definition files associated with source files for
130 the work, and the source code for shared libraries and dynamically
131 linked subprograms that the work is specifically designed to require,
132 such as by intimate data communication or control flow between those
133 subprograms and other parts of the work.

134
135 The Corresponding Source need not include anything that users
136 can regenerate automatically from other parts of the Corresponding
137 Source.

138
139 The Corresponding Source for a work in source code form is that
140 same work.

141 142 2. Basic Permissions.

143
144 All rights granted under this License are granted for the term of
145 copyright on the Program, and are irrevocable provided the stated
146 conditions are met. This License explicitly affirms your unlimited
147 permission to run the unmodified Program. The output from running a
148 covered work is covered by this License only if the output, given its
149 content, constitutes a covered work. This License acknowledges your
150 rights of fair use or other equivalent, as provided by copyright law.

151
152 You may make, run and propagate covered works that you do not
153 convey, without conditions so long as your license otherwise remains
154 in force. You may convey covered works to others for the sole purpose
155 of having them make modifications exclusively for you, or provide you
156 with facilities for running those works, provided that you comply with
157 the terms of this License in conveying all material for which you do
158 not control copyright. Those thus making or running the covered works
159 for you must do so exclusively on your behalf, under your direction
160 and control, on terms that prohibit them from making any copies of
161 your copyrighted material outside their relationship with you.

162
163 Conveying under any other circumstances is permitted solely under
164 the conditions stated below. Sublicensing is not allowed; section 10
165 makes it unnecessary.

166 167 3. Protecting Users' Legal Rights From Anti-Circumvention Law.

168
169 No covered work shall be deemed part of an effective technological
170 measure under any applicable law fulfilling obligations under article
171 11 of the WIPO copyright treaty adopted on 20 December 1996, or
172 similar laws prohibiting or restricting circumvention of such
173 measures.

174
175 When you convey a covered work, you waive any legal power to forbid
176 circumvention of technological measures to the extent such circumvention
177 is effected by exercising rights under this License with respect to

178 the covered work, and you disclaim any intention to limit operation or
179 modification of the work as a means of enforcing, against the work's
180 users, your or third parties' legal rights to forbid circumvention of
181 technological measures.

182

183 4. Conveying Verbatim Copies.

184

185 You may convey verbatim copies of the Program's source code as you
186 receive it, in any medium, provided that you conspicuously and
187 appropriately publish on each copy an appropriate copyright notice;
188 keep intact all notices stating that this License and any
189 non-permissive terms added in accord with section 7 apply to the code;
190 keep intact all notices of the absence of any warranty; and give all
191 recipients a copy of this License along with the Program.

192

193 You may charge any price or no price for each copy that you convey,
194 and you may offer support or warranty protection for a fee.

195

196 5. Conveying Modified Source Versions.

197

198 You may convey a work based on the Program, or the modifications to
199 produce it from the Program, in the form of source code under the
200 terms of section 4, provided that you also meet all of these conditions:

201

202 a) The work must carry prominent notices stating that you modified
203 it, and giving a relevant date.

204

205 b) The work must carry prominent notices stating that it is
206 released under this License and any conditions added under section
207 7. This requirement modifies the requirement in section 4 to
208 "keep intact all notices".

209

210 c) You must license the entire work, as a whole, under this
211 License to anyone who comes into possession of a copy. This
212 License will therefore apply, along with any applicable section 7
213 additional terms, to the whole of the work, and all its parts,
214 regardless of how they are packaged. This License gives no
215 permission to license the work in any other way, but it does not
216 invalidate such permission if you have separately received it.

217

218 d) If the work has interactive user interfaces, each must display
219 Appropriate Legal Notices; however, if the Program has interactive
220 interfaces that do not display Appropriate Legal Notices, your
221 work need not make them do so.

222

223 A compilation of a covered work with other separate and independent
224 works, which are not by their nature extensions of the covered work,
225 and which are not combined with it such as to form a larger program,
226 in or on a volume of a storage or distribution medium, is called an
227 "aggregate" if the compilation and its resulting copyright are not
228 used to limit the access or legal rights of the compilation's users
229 beyond what the individual works permit. Inclusion of a covered work
230 in an aggregate does not cause this License to apply to the other
231 parts of the aggregate.

232

233 6. Conveying Non-Source Forms.

234

235 You may convey a covered work in object code form under the terms
236 of sections 4 and 5, provided that you also convey the

237 machine-readable Corresponding Source under the terms of `this` License,
238 in one of these ways:

239

240 a) Convey the object code in, or embodied in, a physical `product`
241 (including a physical distribution medium), accompanied by the
242 Corresponding Source fixed on a durable physical medium
243 customarily used `for` software interchange.

244

245 b) Convey the object code in, or embodied in, a physical `product`
246 (including a physical distribution medium), accompanied by a
247 written offer, valid `for` at least three years and valid `for` as
248 `long` as you offer spare parts or customer support `for` that product
249 model, to give anyone who possesses the object code `either` (1) a
250 copy of the Corresponding Source `for` all the software in the
251 product that is covered by `this` License, on a durable physical
252 medium customarily used `for` software interchange, `for` a price no
253 more than your reasonable cost of physically performing `this`
254 conveying of source, or (2) access to copy the
255 Corresponding Source from a network server at no charge.

256

257 c) Convey individual copies of the object code with a copy of the
258 written offer to provide the Corresponding Source. This
259 alternative is allowed only occasionally and noncommercially, and
260 only `if` you received the object code with such an offer, in accord
261 with subsection 6b.

262

263 d) Convey the object code by offering access from a designated
264 `place` (gratis or `for` a charge), and offer equivalent access to the
265 Corresponding Source in the same way through the same place at no
266 further charge. You need not require recipients to copy the
267 Corresponding Source along with the object code. If the place to
268 copy the object code is a network server, the Corresponding Source
269 may be on a different `server` (operated by you or a third party)
270 that supports equivalent copying facilities, provided you maintain
271 clear directions next to the object code saying where to find the
272 Corresponding Source. Regardless of what server hosts the
273 Corresponding Source, you remain obligated to ensure that it is
274 available `for` as `long` as needed to satisfy these requirements.

275

276 e) Convey the object code using peer-to-peer transmission, provided
277 you inform other peers where the object code and Corresponding
278 Source of the work are being offered to the general `public` at no
279 charge under subsection 6d.

280

281 A separable portion of the object code, whose source code is excluded
282 from the Corresponding Source as a System Library, need not be
283 included in conveying the object code work.

284

285 A "`User Product`" is `either` (1) a "`consumer product`", which means any
286 tangible personal property which is normally used `for` personal, family,
287 or household purposes, or (2) anything designed or sold `for` incorporation
288 into a dwelling. In determining whether a product is a consumer product,
289 doubtful cases shall be resolved in favor of coverage. For a particular
290 product received by a particular user, "`normally used`" refers to a
291 typical or common use of that `class of` product, regardless of the status
292 of the particular user or of the way in which the particular user
293 actually uses, or expects or is expected to use, the product. A product
294 is a consumer product regardless of whether the product has substantial
295 commercial, industrial or non-consumer uses, unless such uses represent

296 the only significant mode of use of the product.

297

298 "Installation Information" for a User Product means any methods,
 299 procedures, authorization keys, or other information required to install
 300 and execute modified versions of a covered work in that User Product from
 301 a modified version of its Corresponding Source. The information must
 302 suffice to ensure that the continued functioning of the modified object
 303 code is in no case prevented or interfered with solely because
 304 modification has been made.

305

306 If you convey an object code work under this section in, or with, or
 307 specifically for use in, a User Product, and the conveying occurs as
 308 part of a transaction in which the right of possession and use of the
 309 User Product is transferred to the recipient in perpetuity or for a
 310 fixed term (regardless of how the transaction is characterized), the
 311 Corresponding Source conveyed under this section must be accompanied
 312 by the Installation Information. But this requirement does not apply
 313 if neither you nor any third party retains the ability to install
 314 modified object code on the User Product (for example, the work has
 315 been installed in ROM).

316

317 The requirement to provide Installation Information does not include a
 318 requirement to continue to provide support service, warranty, or updates
 319 for a work that has been modified or installed by the recipient, or for
 320 the User Product in which it has been modified or installed. Access to a
 321 network may be denied when the modification itself materially and
 322 adversely affects the operation of the network or violates the rules and
 323 protocols for communication across the network.

324

325 Corresponding Source conveyed, and Installation Information provided,
 326 in accord with this section must be in a format that is publicly
 327 documented (and with an implementation available to the public in
 328 source code form), and must require no special password or key for
 329 unpacking, reading or copying.

330

331 7. Additional Terms.

332

333 "Additional permissions" are terms that supplement the terms of this
 334 License by making exceptions from one or more of its conditions.
 335 Additional permissions that are applicable to the entire Program shall
 336 be treated as though they were included in this License, to the extent
 337 that they are valid under applicable law. If additional permissions
 338 apply only to part of the Program, that part may be used separately
 339 under those permissions, but the entire Program remains governed by
 340 this License without regard to the additional permissions.

341

342 When you convey a copy of a covered work, you may at your option
 343 remove any additional permissions from that copy, or from any part of
 344 it. (Additional permissions may be written to require their own
 345 removal in certain cases when you modify the work.) You may place
 346 additional permissions on material, added by you to a covered work,
 347 for which you have or can give appropriate copyright permission.

348

349 Notwithstanding any other provision of this License, for material you
 350 add to a covered work, you may (if authorized by the copyright holders of
 351 that material) supplement the terms of this License with terms:

352

353 a) Disclaiming warranty or limiting liability differently from the
 354 terms of sections 15 and 16 of this License; or

- b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or
- c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or
- d) Limiting the use **for** publicity purposes of names of licensors or authors of the material; or
- e) Declining to grant rights under trademark law **for** use of some trade names, trademarks, or service marks; or
- f) Requiring indemnification of licensors and authors of that material by anyone who conveys the **material** (or modified versions of it) with contractual assumptions of liability to the recipient, **for** any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered "**further restrictions**" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by **this** License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under **this** License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with **this** section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

You may not propagate or modify a covered work except as expressly provided under **this** License. Any attempt otherwise to propagate or modify it is **void**, and will automatically terminate your rights under **this** License (including any patent licenses granted under the third paragraph of section 11).

However, **if** you cease all violation of **this** License, then your license from a particular copyright holder is **reinstated** (a) provisionally, unless and until the copyright holder explicitly and **finally** terminates your license, and (b) permanently, **if** the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently **if** the copyright holder notifies you of the violation by some reasonable means, **this** is the first time you have received notice of violation of **this** License (**for** any work) from that

414 copyright holder, and you cure the violation prior to 30 days after
415 your receipt of the notice.

416
417 Termination of your rights under **this** section does not terminate the
418 licenses of parties who have received copies or rights from you under
419 **this** License. If your rights have been terminated and not permanently
420 reinstated, you **do** not qualify to receive **new** licenses **for** the same
421 material under section 10.

422 423 9. Acceptance Not Required **for** Having Copies.

424
425 You are not required to accept **this** License in order to receive or
426 run a copy of the Program. Ancillary propagation of a covered work
427 occurring solely as a consequence of using peer-to-peer transmission
428 to receive a copy likewise does not require acceptance. However,
429 nothing other than **this** License grants you permission to propagate or
430 modify any covered work. These actions infringe copyright **if** you **do**
431 not accept **this** License. Therefore, by modifying or propagating a
432 covered work, you indicate your acceptance of **this** License to **do** so.

433 434 10. Automatic Licensing of Downstream Recipients.

435
436 Each time you convey a covered work, the recipient automatically
437 receives a license from the original licensors, to run, modify and
438 propagate that work, subject to **this** License. You are not responsible
439 **for** enforcing compliance by third parties with **this** License.

440
441 An "**entity transaction**" is a transaction transferring control of an
442 organization, or substantially all assets of one, or subdividing an
443 organization, or merging organizations. If propagation of a covered
444 work results from an entity transaction, each party to that
445 transaction who receives a copy of the work also receives whatever
446 licenses to the work the party's predecessor in interest had or could
447 give under the previous paragraph, plus a right to possession of the
448 Corresponding Source of the work from the predecessor in interest, **if**
449 the predecessor has it or can get it with reasonable efforts.

450
451 You may not impose any further restrictions on the exercise of the
452 rights granted or affirmed under **this** License. For example, you may
453 not impose a license fee, royalty, or other charge **for** exercise of
454 rights granted under **this** License, and you may not initiate **litigation**
455 (including a cross-claim or counterclaim in a lawsuit) alleging that
456 any patent claim is infringed by making, using, selling, offering **for**
457 sale, or importing the Program or any portion of it.

458 459 11. Patents.

460
461 A "**contributor**" is a copyright holder who authorizes use under **this**
462 License of the Program or a work on which the Program is based. The
463 work thus licensed is called the contributor's "**contributor version**".

464
465 A contributor's "**essential patent claims**" are all patent claims
466 owned or controlled by the contributor, whether already acquired or
467 hereafter acquired, that would be infringed by some manner, permitted
468 by **this** License, of making, using, or selling its contributor version,
469 but **do** not include claims that would be infringed only as a
470 consequence of further modification of the contributor version. For
471 purposes of **this** definition, "**control**" includes the right to grant
472 patent sublicenses in a manner consistent with the requirements of

473 this License.

474

475 Each contributor grants you a non-exclusive, worldwide, royalty-free
476 patent license under the contributor's essential patent claims, to
477 make, use, sell, offer for sale, import and otherwise run, modify and
478 propagate the contents of its contributor version.

479

480 In the following three paragraphs, a "patent license" is any express
481 agreement or commitment, however denominated, not to enforce a patent
482 (such as an express permission to practice a patent or covenant not to
483 sue for patent infringement). To "grant" such a patent license to a
484 party means to make such an agreement or commitment not to enforce a
485 patent against the party.

486

487 If you convey a covered work, knowingly relying on a patent license,
488 and the Corresponding Source of the work is not available for anyone
489 to copy, free of charge and under the terms of this License, through a
490 publicly available network server or other readily accessible means,
491 then you must either (1) cause the Corresponding Source to be so
492 available, or (2) arrange to deprive yourself of the benefit of the
493 patent license for this particular work, or (3) arrange, in a manner
494 consistent with the requirements of this License, to extend the patent
495 license to downstream recipients. "Knowingly relying" means you have
496 actual knowledge that, but for the patent license, your conveying the
497 covered work in a country, or your recipient's use of the covered work
498 in a country, would infringe one or more identifiable patents in that
499 country that you have reason to believe are valid.

500

501 If, pursuant to or in connection with a single transaction or
502 arrangement, you convey, or propagate by procuring conveyance of, a
503 covered work, and grant a patent license to some of the parties
504 receiving the covered work authorizing them to use, propagate, modify
505 or convey a specific copy of the covered work, then the patent license
506 you grant is automatically extended to all recipients of the covered
507 work and works based on it.

508

509 A patent license is "discriminatory" if it does not include within
510 the scope of its coverage, prohibits the exercise of, or is
511 conditioned on the non-exercise of one or more of the rights that are
512 specifically granted under this License. You may not convey a covered
513 work if you are a party to an arrangement with a third party that is
514 in the business of distributing software, under which you make payment
515 to the third party based on the extent of your activity of conveying
516 the work, and under which the third party grants, to any of the
517 parties who would receive the covered work from you, a discriminatory
518 patent license (a) in connection with copies of the covered work
519 conveyed by you (or copies made from those copies), or (b) primarily
520 for and in connection with specific products or compilations that
521 contain the covered work, unless you entered into that arrangement,
522 or that patent license was granted, prior to 28 March 2007.

523

524 Nothing in this License shall be construed as excluding or limiting
525 any implied license or other defenses to infringement that may
526 otherwise be available to you under applicable patent law.

527

528 12. No Surrender of Others' Freedom.

529

530 If conditions are imposed on you (whether by court order, agreement or
531 otherwise) that contradict the conditions of this License, they do not

excuse you from the conditions of `this` License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under `this` License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, `if` you agree to terms that obligate you to collect a royalty `for` further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and `this` License would be to refrain entirely from conveying the Program.

13. Remote Network Interaction; Use with the GNU General Public License.

Notwithstanding any other provision of `this` License, `if` you modify the Program, your modified version must prominently offer all users interacting with it remotely through a computer `network` (`if` your version supports such interaction) an opportunity to receive the Corresponding Source of your version by providing access to the Corresponding Source from a network server at no charge, through some standard or customary means of facilitating copying of software. This Corresponding Source shall include the Corresponding Source `for` any work covered by version 3 of the GNU General Public License that is incorporated pursuant to the following paragraph.

Notwithstanding any other provision of `this` License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU General Public License into a single combined work, and to convey the resulting work. The terms of `this` License will `continue` to apply to the part which is the covered work, but the work with which it is combined will remain governed by version 3 of the GNU General Public License.

14. Revised Versions of `this` License.

The Free Software Foundation may publish revised and/or `new` versions of the GNU Affero General Public License from time to time. Such `new` versions will be similar in spirit to the present version, but may differ in detail to address `new` problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU Affero General Public License "`or any later version`" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU Affero General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU Affero General Public License can be used, that proxy's `public` statement of acceptance of a version permanently authorizes you to choose that version `for` the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT

591 HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY
592 OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO,
593 THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
594 PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM
595 IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF
596 ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

597
598 16. Limitation of Liability.

599
600 IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING
601 WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS
602 THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY
603 GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE
604 USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF
605 DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD
606 PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS),
607 EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF
608 SUCH DAMAGES.

609
610 17. Interpretation of Sections 15 and 16.

611
612 If the disclaimer of warranty and limitation of liability provided
613 above cannot be given local legal effect according to their terms,
614 reviewing courts shall apply local law that most closely approximates
615 an absolute waiver of all civil liability in connection with the
616 Program, unless a warranty or assumption of liability accompanies a
617 copy of the Program in return for a fee.

618
619 END OF TERMS AND CONDITIONS

620
621 How to Apply These Terms to Your New Programs

622
623 If you develop a new program, and you want it to be of the greatest
624 possible use to the public, the best way to achieve this is to make it
625 free software which everyone can redistribute and change under these terms.

626
627 To do so, attach the following notices to the program. It is safest
628 to attach them to the start of each source file to most effectively
629 state the exclusion of warranty; and each file should have at least
630 the "copyright" line and a pointer to where the full notice is found.

631
632 <one line to give the program's name and a brief idea of what it does.>
633 Copyright (C) <year> <name of author>

634
635 This program is free software: you can redistribute it and/or modify
636 it under the terms of the GNU Affero General Public License as published
637 by the Free Software Foundation, either version 3 of the License, or
638 (at your option) any later version.

639
640 This program is distributed in the hope that it will be useful,
641 but WITHOUT ANY WARRANTY; without even the implied warranty of
642 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
643 GNU Affero General Public License for more details.

644
645 You should have received a copy of the GNU Affero General Public License
646 along with this program. If not, see <<http://www.gnu.org/licenses/>>.

647
648 Also add information on how to contact you by electronic and paper mail.

649

650 If your software can interact with users remotely through a computer
 651 network, you should also make sure that it provides a way for users to
 652 get its source. For example, if your program is a web application, its
 653 interface could display a "Source" link that leads users to an archive
 654 of the code. There are many ways you could offer source, and different
 655 solutions will be better for different programs; see section 13 for the
 656 specific requirements.

657

658 You should also get your employer (if you work as a programmer) or school,
 659 if any, to sign a "copyright disclaimer" for the program, if necessary.
 660 For more information on this, and how to apply and follow the GNU AGPL, see
 661 <<http://www.gnu.org/licenses/>>.

README.md

```

1  # COBrA
2  An Ethereum blockchain university project.
3
4  For more information heck the assignments for [COBrA] (docs/assignment/COBrA_Assignment.pdf) and
5  [COBrA DAPP] (docs/assignment/COBrA_DAPP_Assignment.pdf) and the
6  [COBrA DAPP Relationship] (docs/COBrA_DAPP_Relationship.pdf).
7
8  See also the [Javadoc] (https://daquinoaldo.github.io/COBrA/javadoc).
9
10 ## Requirements
11 - [Ethereum (geth)] (https://geth.ethereum.org/downloads/)
12   - Ubuntu:
13     - `sudo add-apt-repository -y ppa:ethereum/ethereum`
14     - `sudo apt-get update`
15     - `sudo apt-get install ethereum`
16   - MacOS:
17     - `brew tap ethereum/ethereum`
18     - `brew install ethereum`
19   - Compile sources (requires [Go] (https://golang.org/dl/)):
20     - `go install github.com/ethereum/go-ethereum/cmd/geth`
21 - [Solidity compiler] (https://github.com/ethereum/solidity)
22   - Ubuntu:
23     - `sudo add-apt-repository ppa:ethereum/ethereum`
24     - `sudo apt-get update`
25     - `sudo apt-get install solc`
26   - MacOS:
27     - `brew tap ethereum/ethereum`
28     - `brew install solidity`
29   - [Build from
30     ↪ sources] (http://solidity.readthedocs.io/en/v0.4.24/installing-solidity.html#building-from-source)
31 - Java 10
32   - ↪ [JRE] (http://www.oracle.com/technetwork/java/javase/downloads/jre10-downloads-4417026.html)
33   - ↪ [JDK] (http://www.oracle.com/technetwork/java/javase/downloads/jdk10-downloads-4416644.html)
34 - [Apache Maven] (https://maven.apache.org/install.html)
35   - Ubuntu: `sudo apt-get install maven`
36   - MacOS: `sudo brew install maven` (requires [brew] (https://docs.brew.sh/Installation))
37   - [Binaries] (https://maven.apache.org/download.cgi)
38 ## Run
39 ### Compile sources

```



```

40 To compile solidity contracts, generate Java contract with web3j and build JARs run bash
    ↳ install.sh.
41
42 ### Ethereum client
43 Start and Ethereum node on the testnet Ropsten with
    ↳ [geth] (https://github.com/ethereum/go-ethereum/wiki/geth) running
44 geth --rpcapi personal,db,eth,net,web3 --rpc --testnet or start an emulated node with
    ↳ ganache-cli.
45
46 ### App
47 Run the GUI with java -jar DAPP/jar/gui-1.0-jar-with-dependencies.jar. It starts a wizard
    ↳ that allows you to create
48 credentials from your private key, deploy a new catalog or connect to an existent one and
    ↳ choose if you want the
49 author's or the customer's view.
50
51 Authors also need a running author-server in which store their content in order to deploy a
    ↳ ContentManagementContract.
52 You can run an author-server with java -jar
    ↳ DAPP/jar/author-server-1.0-jar-with-dependencies.jar -k <your-private-key>
53 -c <existent-catalog-contract-address>.
54 The contract deploy can be done from the author's GUI, then the GUI can be stopped and the
    ↳ content will remain available
55 as far as the author-server remain online.

```