

I3 User Documentation

Version 0

04-August-2019

Overview of I3

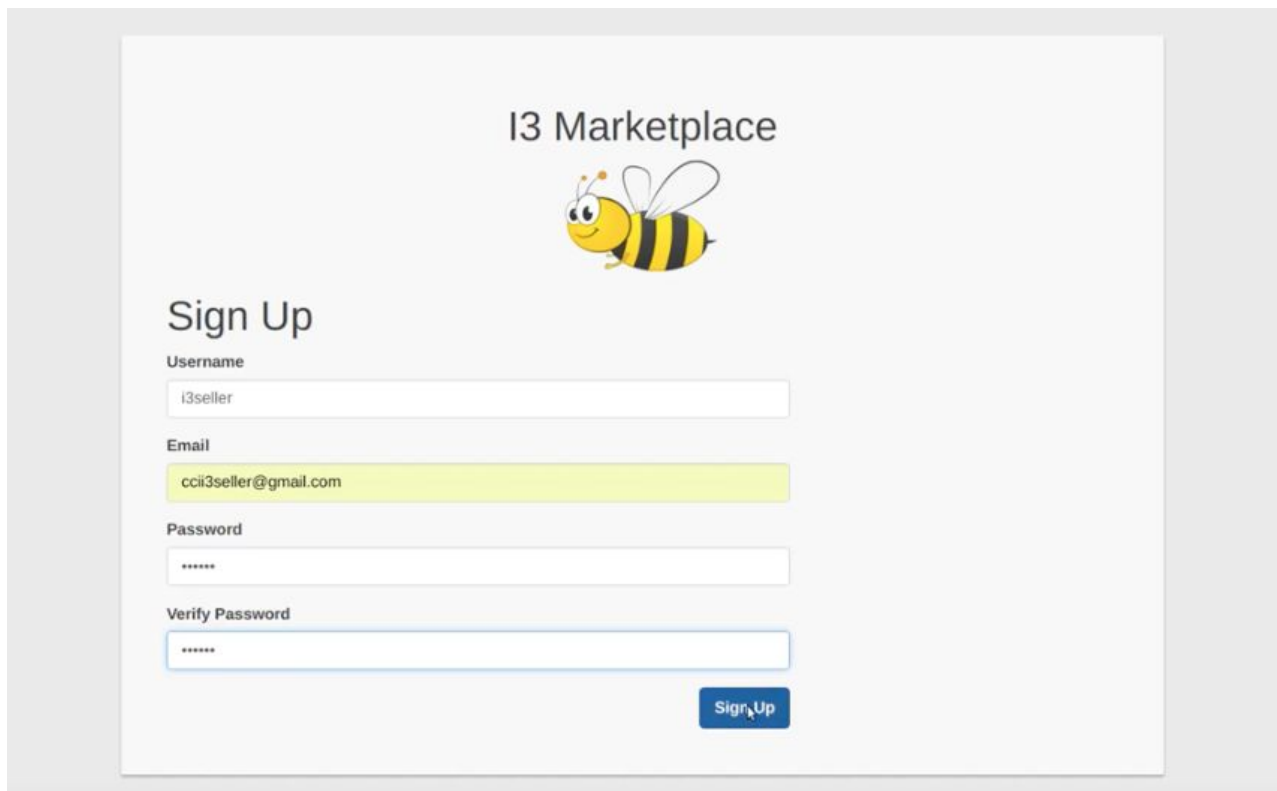
The goal of the I3 system is to create IoT communities including smart cities, where device owners are an active part of the community. These IoT communities can link themselves together to create larger communities. Applications and data brokers access and provide value-added services to these IOT-based communities. The I3 consortium is an open public-private partnership working towards this goal.

This document explains how buyers and sellers can use the I3 marketplace (Version 0). Currently, the implementation supports a web frontend and REST APIs for the users.

I3 Marketplace Frontend GUI

1. Creating an account

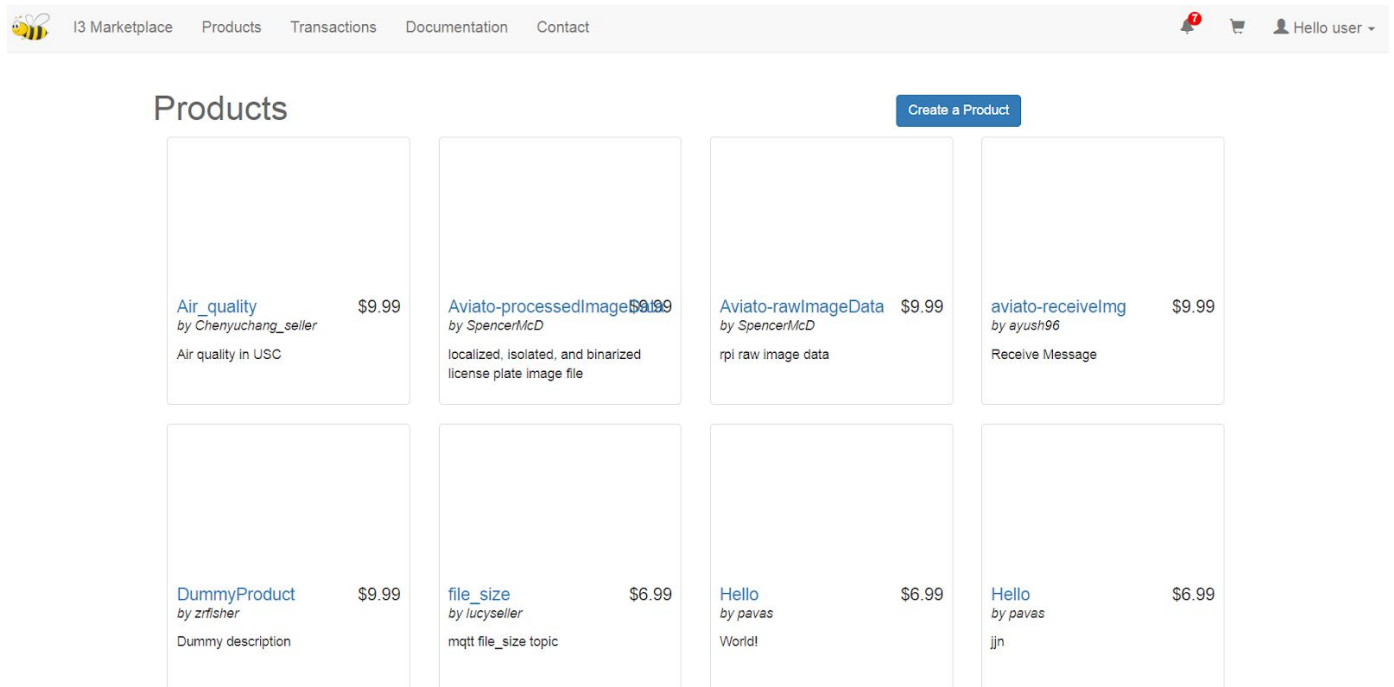
Once you access the login page, sign up and create an account. In the figure below, we are going to create a 'seller' account.



The screenshot displays the 'I3 Marketplace' sign-up interface. At the top center, the text 'I3 Marketplace' is positioned above a cartoon bee logo. Below this, the heading 'Sign Up' is followed by four input fields: 'Username' (containing 'i3seller'), 'Email' (containing 'ccii3seller@gmail.com'), 'Password' (masked with dots), and 'Verify Password' (also masked with dots). A blue 'Sign Up' button is located at the bottom right of the form area.

2. Creating a product

To create a product, navigate to the Product Section and click on the Create Product button which would navigate you to Create New Product Page. The Product page is shown below

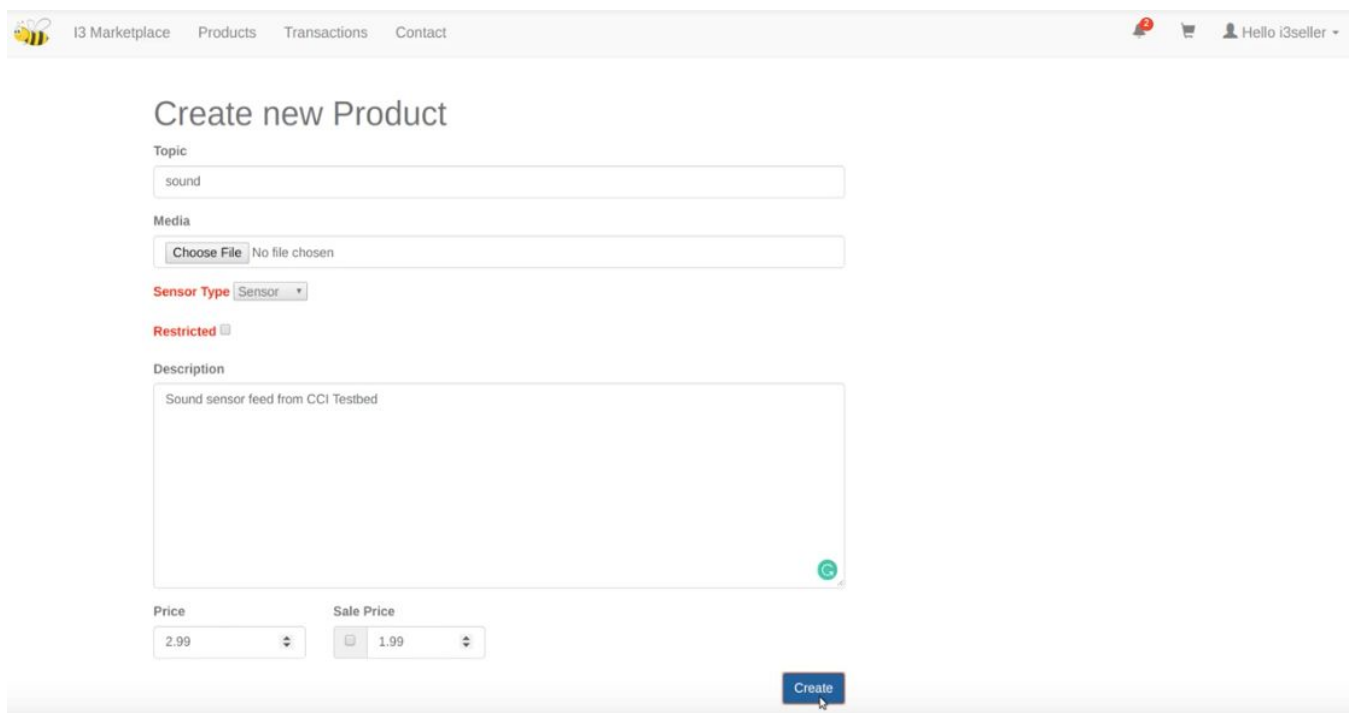


The screenshot shows the 'Products' page of the I3 Marketplace. The header includes a navigation bar with 'I3 Marketplace', 'Products', 'Transactions', 'Documentation', and 'Contact'. On the right, there are icons for notifications, a shopping cart, and a user profile labeled 'Hello user'. A 'Create a Product' button is located in the top right corner of the product grid.

The product grid displays eight items in a 2x4 layout:

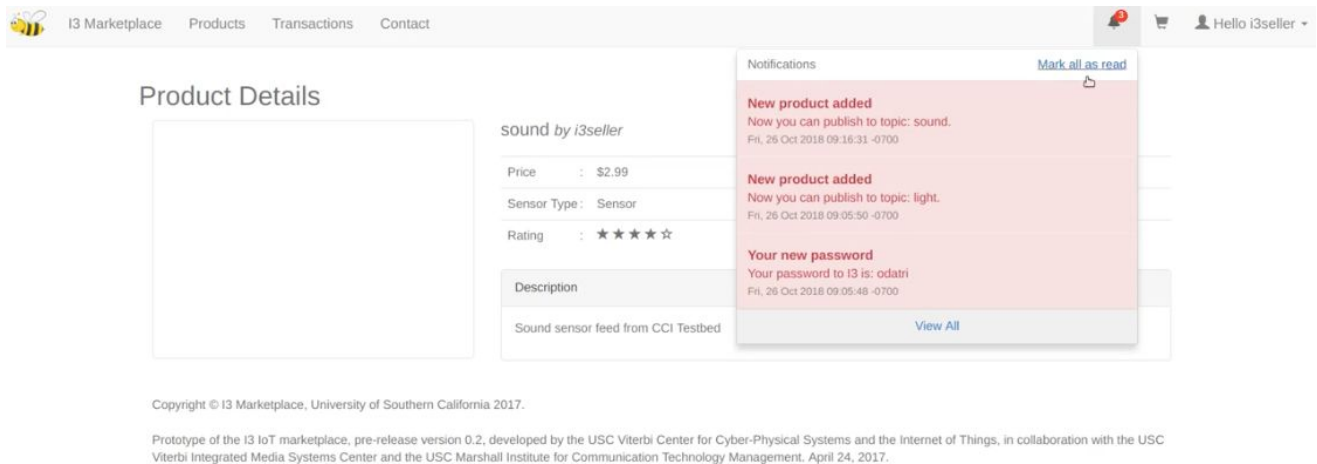
- Air_quality** by Chenyuchang_seller, \$9.99, Description: Air quality in USC
- Aviato-processedImage** by SpencerMcD, \$9.99, Description: localized, isolated, and binarized license plate image file
- Aviato-rawImageData** by SpencerMcD, \$9.99, Description: rpi raw image data
- aviato-receiveImg** by ayush96, \$9.99, Description: Receive Message
- DummyProduct** by zrfisher, \$9.99, Description: Dummy description
- file_size** by lucyseller, \$6.99, Description: mqtt file_size topic
- Hello** by pavas, \$6.99, Description: World!
- Hello** by pavas, \$6.99, Description: iijn

On the create new product page specify the necessary details (Topic, Media, Sensor Type, Description, etc.) of your product as shown in the above example.



The screenshot shows the 'Create new Product' form. The header is identical to the previous page, but the user profile is labeled 'Hello i3seller'. The form fields are as follows:

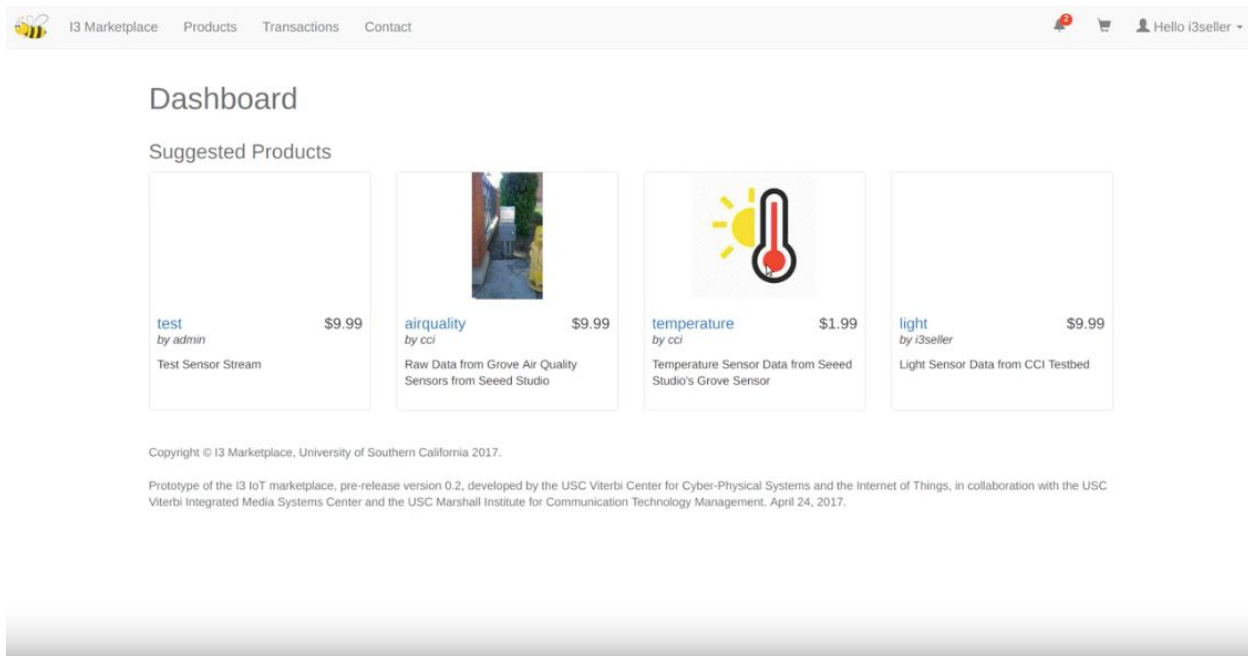
- Topic:** A text input field containing 'sound'.
- Media:** A file upload section with a 'Choose File' button and the text 'No file chosen'.
- Sensor Type:** A dropdown menu currently set to 'Sensor'.
- Restricted:** A checkbox that is currently unchecked.
- Description:** A large text area containing 'Sound sensor feed from CCI Testbed'.
- Price:** A numeric input field set to '2.99'.
- Sale Price:** A numeric input field set to '1.99'.
- Create:** A blue button at the bottom right of the form.



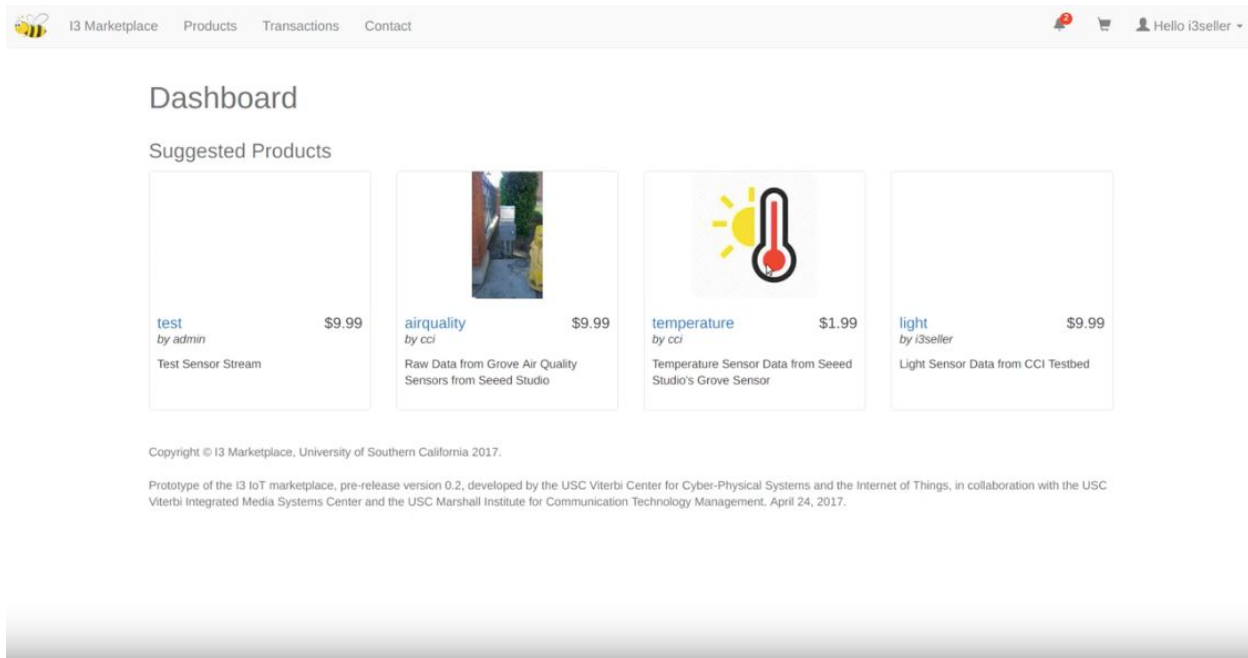
Once you have successfully created your product, you will be notified of your publishing credentials. These credentials will allow sensors to ‘publish’ to the topic specified. In the above example, we have obtained credentials (password) to publish to the topic ‘sound’.

Note: The system generates a password for you, for the first time when you buy/create a product. This password is expected to be used for all communication with the MQTT broker. The password appears in the notifications.

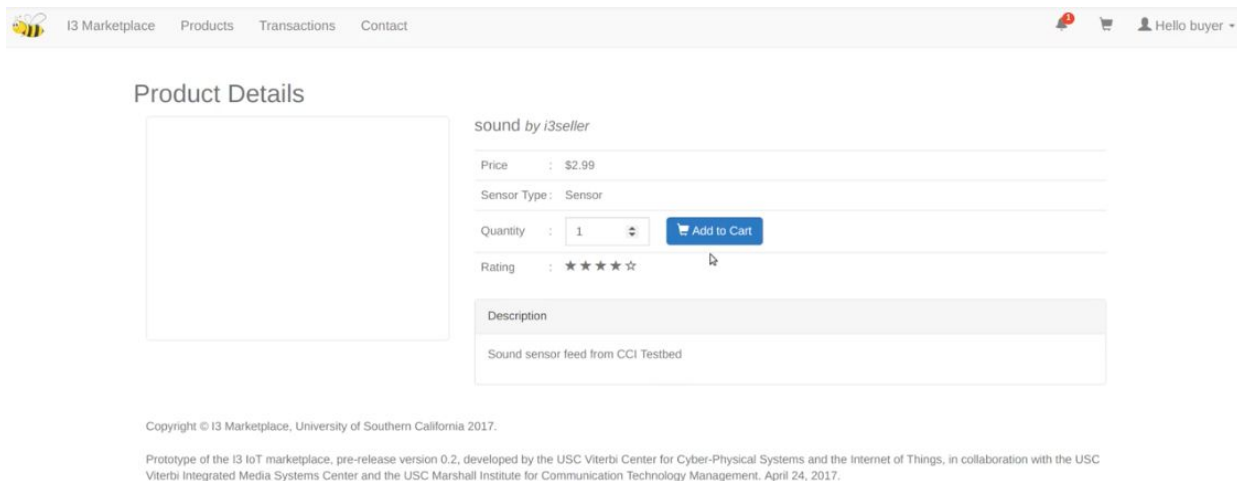
3. Buying a product



Create an account in the system using the instructions in Step 1, if you don't have an account in the system already. After successfully logging in as a Buyer, browse through the listed products and select a Product by clicking on their topic from the dashboard.



Purchase the product by selecting 'Add to Cart'. This will direct you to a payment gateway to complete your purchase.



Once the product is successfully purchased, it will appear as a transaction under 'Purchases'. In the notifications tab, you will also receive confirmation that you are allowed to subscribe to the products that were purchased.

Note: The system generates a password for you, for the first time when you buy/create a product. This password is expected to be used for all communication with the MQTT broker. The password appears in the notifications.

4. Publish data from IoT Device

I3 marketplace uses a publish-subscribe communication model over MQTT and Web Sockets. The seller has to set up his IoT device with the registered topic, username, and password [system generated - can be found under notifications] to publish the data to I3 broker. MQTT libraries are available in multiple programming languages, including Python, Java, JavaScript, C, etc. You can find an example Python script at the following link: [Publish.Py script](#). For testing please use the following broker configuration to publish to the marketplace broker:

- Broker address: 3.15.198.123
- Broker port: 1883
- Use the username and password [system generated] that you obtained from the marketplace
- The topic is the name of the product that you purchased from the marketplace

The data can also be published with transport layer encryption enabled. You can find an example Python script at the following link: [TLS Enabled publish.py script](#). For testing please use the following broker configuration to publish to the marketplace broker:

- Broker address: 3.15.198.123
- Broker port: 8883 for TLS 1.2 and 9001 for TLS 1.1
- Use the username and password [system generated] that you obtained from the marketplace
- The topic is the name of the product that you purchased from the marketplace

The instance also supports web socket connection on port 9010.

5. Subscribing to data from IoT Device

When you have purchased a data product through the I3 marketplace, you will be given access credentials, which include the topic, username, and password [system generated - can be found under notifications]. The buyer has to configure his client software with appropriate credentials to receive the data from the seller. MQTT libraries are available in multiple programming languages, including Python, Java, JavaScript, C, etc. You can find an example Python script at the following link: [Subscribe.Py script](#). For testing please use the following broker configuration to subscribe from the marketplace broker:

- Broker address: 3.15.198.123
- Broker port: 1883
- Use the username and password [system generated] that you obtained from the marketplace
- The topic is the name of the product that you purchased from the marketplace

The data can also be with transport layer encryption enabled. You can find an example Python script at the following link: [TLS Enabled subscribe.py script](#). For testing please use the following broker configuration to subscribe to the marketplace broker:

- Broker address: 3.15.198.123
- Broker port: 8883 for TLS 1.2 and 9001 for TLS 1.1
- Use the username and password [system generated] that you obtained from the marketplace
- The topic is the name of the product that you purchased from the marketplace

The instance also supports web socket connection on port 9010.

I3 Marketplace Frontend REST API

1. Description of the API

The I3 marketplace has a REST API to access all the fundamental features of the frontend. All fundamental features of the frontend can be utilized with CRUD operations – create, retrieve, update, destroy. A user can access the API and perform some or all CRUD operations based on what API endpoints they are accessing.

2. Usage of the API

a. Authorization

A user must obtain an Authorization Token from the I3 marketplace in order to gain access to the API. One can access the token generation page at **/key** or by clicking the link from the dropdown on the navbar. Once the key has been obtained, this key must be used in every API request sent by the user. The key should be sent in the headers of the request. The key should be sent using the **Authorization** header.

Example : “**Authorization: Token 085d75216d6716bfac72d9a5b76d6bf9c7525c12**”

b. Outline

Here is an outline of all API endpoints and their descriptions. In some of the “**Descriptions of Methods**”, “**cURL**” is using a **cURL** is a command-line tool that can transfer data from or to a server using many different protocols. In our case, we will use **cURL** and the HTTP protocol to interface with the API. The API is also broken into two use cases. One is for a *buyer* and one is for a *seller*. Buyers and sellers have different actions so they have two different routes to access the API.

URL, Accepted Methods and Relevant Info	Description of Methods
/docs	A documentation page for the API including descriptions of each endpoint as well as useful code segments. It is a supplement to this document.
/key	Page to acquire users API key needed for every request.
BUYER	

/api/buyer/product/

GET:

Query Parameters	Description
<i>offset</i>	Amount to offset the beginning of the list of products. The default is 0.
<i>limit</i>	Amount of items to return. The default is 10.
<i>sensor_type</i>	Filter by sensor type. Must be <ul style="list-style-type: none">• sensor• actuator• both
<i>seller</i>	Filter by seller

GET:

Returns an array of all products on the marketplace.

cURL Example:

```
user$ curl -X GET -H 'Content
Type: application/json' -H
'Authorization: Token
402f1c300e9f67652a3cfa9f407fca
b154e6c026'
http://3.15.198.123:8000/api/b
uyer/product/?offset=0&sensor_
type=sensor&seller=admin
```

/api/buyer/product/<id>

GET:

Path Variable	Description
<id>	The ID of the product to access

GET:

Returns a specific product based on the id.

cURL Example:

```
user$ curl -X GET -H 'Content
Type: application/json' -H
'Authorization: Token
402f1c300e9f67652a3cfa9f407fca
b154e6c026'
http://3.15.198.123:8000/api/b
uyer/product/1/
```

/api/buyer/purchase

GET:

GET:

Returns a list of all purchases made by the user.

cURL Example:

<p>POST: Example JSON to be submitted with the POST request.</p> <pre>//test.json { "id" : "4", "quantity" : "3" }</pre>	<pre>user\$ curl -X GET -H 'Content Type: application/json' -H 'Authorization: Token 402f1c300e9f67652a3cfa9f407fca b154e6c026' http://3.15.198.123:8000/api/b uyer/purchase/</pre> <p>POST: Purchase a product based on the product id.</p> <p>cURL Example:</p> <pre>user\$ curl -X POST -H 'Content Type: application/json' -H 'Authorization: Token 402f1c300e9f67652a3cfa9f407fca b154e6c026' -d '{"id":"1","quantity":"3"}' http://3.15.198.123:8000/api/b uyer/purchase/</pre>				
<p>/api/buyer/purchase/<id></p> <p>GET:</p> <table border="1" data-bbox="326 1459 889 1633"> <thead> <tr> <th>Parameter</th><th>Description</th></tr> </thead> <tbody> <tr> <td><id></td><td>ID of product to access</td></tr> </tbody> </table>	Parameter	Description	<id>	ID of product to access	<p>GET: Return a specific purchase made by the user based on the purchase id.</p> <p>cURL Example:</p> <pre>user\$ curl -X GET -H 'Content Type: application/json' -H 'Authorization: Token 402f1c300e9f67652a3cfa9f407fca b154e6c026' http://3.15.198.123:8000/api/b uyer/purchase/1/</pre>
Parameter	Description				
<id>	ID of product to access				
<p>SELLER</p>					
<p>/api/seller/product/</p>	<p>GET:</p>				

GET:

Query Parameters	Description								
offset	Amount to offset the beginning of the list of products. Default is 0. <table><tr><td>offset=</td><td>Output</td></tr><tr><td>0</td><td>Items 0-9</td></tr><tr><td>1</td><td>Items 1-10</td></tr><tr><td>10</td><td>Items 10-19</td></tr></table>	offset=	Output	0	Items 0-9	1	Items 1-10	10	Items 10-19
offset=	Output								
0	Items 0-9								
1	Items 1-10								
10	Items 10-19								
limit	Amount of items to return. Default is 10.								
sensor_type	Filter by sensor type. Must be <ul style="list-style-type: none">• sensor• actuator• both								
seller	Filter by seller								

POST:

Example JSON to be submitted with POST request

```
//test.json
```

Returns an array of all products on the marketplace.

cURL Example:

```
user$ curl -X GET -H 'Content
Type: application/json' -H
'Authorization: Token
0bc5c14700b96401b244f3568ef935
3f25cda23d'
http://3.15.198.123:8000/api/s
eller/product/?offset=0&sensor
_type=sensor&seller=admin
```

POST:

Create a new product. **POST** request must include the data for the new product in **JSON** format.

cURL Example:

```
user$ curl -X POST -H 'Content
Type: application/json' -H
'Authorization: Token
0bc5c14700b96401b244f3568ef935
3f25cda23d' -d '{"seller":
"admin","media": null,
"title": "test title",
"sensor_type": "sensor",
"description": "test
description", "price": "9.99",
"sale_active": false,
"sale_price": "6.99"}'
http://3.15.198.123:8000/api/s
eller/product/5/
```

```
{
  "media":
    "http://samplemedia.url",
    "title": "sample title",
    "sensor_type": "sensor",
    "description": "sample
description",
    "price": "9.99",
    "sale_active": false,
    "sale_price": "6.99"
}
```

/api/seller/product/<id>

GET:

Path Variable	Description
<id>	ID of product to access

PUT:

Example JSON to be submitted with PUT request.

```
//test.json
{
  "media":
    "http://samplemedia.url",
    "title": "sample title",
    "sensor_type": "sensor",
    "description": "sample
description",
    "price": "9.99",
    "sale_active": false,
    "sale_price": "6.99"
}
```

GET:

Return a specific product based on the product **id**.

cURL Example:

```
user$ curl -X GET -H 'Content
Type: application/json' -H
'Authorization: Token
0bc5c14700b96401b244f3568ef935
3f25cda23d'
http://3.15.198.123:8000/api/s
eller/product/2/
```

PUT:

Update a product based on the product **id**. The authorization token must correspond to the user who created the product in question. A user can only update a product if that user created it.

cURL Example:

```
user$ curl -X PUT -H 'Content
Type: application/json' -H
'Authorization: Token
0bc5c14700b96401b244f3568ef935
3f25cda23d' -d '{"seller":
"admin","media": null,
"title": "test title",
```

<p>DELETE:</p>	<div data-bbox="919 138 1484 474" data-label="Text"> <pre>"sensor_type": "sensor", "description": "test description", "price": "9.99", "sale_active": false, "sale_price": "6.99"}' http://3.15.198.123:8000/api/seller/product/2/</pre> </div> <p>DELETE: Delete a product based on the product id. The authorization token must correspond to the user who created the product in question. A user can only delete a product if that user created it.</p> <p>cURL Example:</p> <div data-bbox="919 852 1484 1234" data-label="Text"> <pre>user\$ curl -X DELETE -H 'Content Type: application/json' -H 'Authorization: Token 0bc5c14700b96401b244f3568ef935 3f25cda23d' http://3.15.198.123:8000/api/seller/product/1/</pre> </div>
<p>/api/seller/sale/ GET:</p>	<p>GET: Return a list of all sales by the user.</p> <p>cURL Example:</p> <div data-bbox="919 1528 1484 1822" data-label="Text"> <pre>user\$ curl -X GET -H 'Content Type: application/json' -H 'Authorization: Token 0bc5c14700b96401b244f3568ef935 3f25cda23d' http://3.15.198.123:8000/api/seller/sale/</pre> </div>
<p>/api/seller/sale/<id></p>	<p>GET: Return a specific sale by the user</p>

GET:		cURL Example: <pre> user\$ curl -X GET -H 'Content Type: application/json' -H 'Authorization: Token 0bc5c14700b96401b244f3568ef935 3f25cda23d' http://3.15.198.123:8000/api/s eller/sale/0/ </pre>
Path Variable	Description	
<id>	ID of sale to access	

c. Example usage

- i. Acquire API key from `http://3.15.198.123:8000/key`
- ii. Place key in headers of request
 1. Example - `'Authorization: Token 085d75216d6716bfac72d9a5b76d6bf9c7525c12'`
- iii. Send request to desired endpoint with attached key