@GET('^/api/v1/get_image')
arguments:
    hash= the hash of the image (40 character hex string)
    guid= the guid of the node to get the image from. should be omitted if retrieving images
        from own node (40 character hex string)
return:
    the image which can be loaded in a browser

@GET('^/api/v1/profile')
arguments:
    guid= the guid of the node to get the profile from. should be omitted if retrieving the
        profile for own node. (40 character hex string)
return:
    the profile as a json object

@GET('^/api/v1/get_listings')
arguments:
    guid= the guid of the node to get the listings from. should be omitted if retrieving the
        listings for own node. (40 character hex string)
return:
    a json object containing a list of listing objects

@GET('^/api/v1/get_followers')
arguments:
    guid= the guid of the node to get the followers for. should be omitted if retrieving the
        followers for own node. (40 character hex string)
return:
    a json object containing a list of follower objects.

@GET('^/api/v1/get_following')
arguments:
    guid= the guid of the node to get the following nodes for. should be omitted if retrieving
        the following for own node. (40 character hex string)
return: a json object containing a list of following objects.

@POST('^/api/v1/follow')
arguments:
    guid= the guid of the node to follow (40 character hex string)
action:
    sends a follow message to the node and causes them to store your guid
    (and some metadata) in their follower database.
return:
    {success: true}

{success: false} if the node was unreachable

@POST('^/api/v1/unfollow')
arguments:
guid= the guid of the node to unfollow (40 character hex string)
action:
sends an unfollow message to the node and causes them to delete your metadata from their followers database. Note: a malicious node could refuse to delete causing you to show as a follower after you unfollow.
return:
{success: true}
{success: false} if the node was unreachable

@POST('^/api/v1/profile')
arguments:
name= the user or store's name (string)
location= the user's location. must be country found in this list and must be formatted the same. Case sensitivity doesn't matter, however. (properly formatted string)

The above two arguments are required protobuf fields. They must be set on the first call to create the profile. They may be omitted on subsequent calls to update the profile. The remaining arguments are optional.

handle= the onename handle starting with "@". eventually this will be required to resolve to the guid. (string starting with @).
about= text to show in the user's 'about' section. (string)
short_description= text to show in the homepage store list (string)
nsfw= Is this user profile/store nsfw? Will default to false if the field is omitted. ("true" or "false")
vendor= is this user a vendor? must be set to true if so. defaults to false if omitted. ("true" or "false")
moderator= is this user a moderator? must be set to true if so. defaults to false if omitted. ("true" or "false")
moderation_fee= the percentage fee changed by the moderator (ex 10.0)
website= a website for this user (string)
email= an email address for this user (string)
primary_color = hex color formatted in base 10. For example, 00FF00 should be sent as "65280" (string of base 10 formatted hex color)
secondary_color= same as primary color
text_color= same as primary color
background_color= same as primary color
avatar= the hash of the avatar image. must have been previously uploaded using the

upload_image api call (40 character hex string)
header= the hash of the header image. must have been previously uploaded using the upload_image api call. (40 character hex string)
pgp_key= a pgp public key to include in the profile. if included the signature field must also be included. (string pgp public key block)
signature= a pgp signature covering the user's guid. must be sent along with the pgp_key field. (string pgp signed message covering the user guid)

action:

sets the profile in the db. will be visible to other nodes.

return:

{success: True}

@POST('^/api/v1/social_accounts')

arguments:

red arguments are mandatory
account_type= must be either "facebook", "twitter", "instagram", or "snapchat" (string)
username= the account user name (string)
proof= a url proving ownership of the social account. this is not validated at the moment (string url)

action:

adds the social account to the user profile

return:

{success: True}

@DELETE('^/api/v1/social_accounts')

arguments:

account_type= the account to delete. must be either "facebook", "twitter", "instagram", or "snapchat" (string)

action:

remove the social account from the profile

return:

{success: True}

@GET('^/api/v1/contracts')

arguments:

id= the contract id to fetch (40 character hex string)
guid= the guid of the node to fetch the contract from. should be omitted if getting contract from own node (40 character hex string)

return:

the contract json object or {} if it couldn't be located

@POST('^/api/v1/contracts')

arguments:

red is mandatory
expiration_date= the date the contract should expire in string formatted UTC datetime. example: "2015-11-01T00:00 UTC" or “” if the contract never expires. (formatted string)
metadata_category= “physical good”, “digital good”, or “service” (formatted string)
title= title of the product for sale (string)
description= description of the product (string)
currency_code= the currency the product is priced in. may either be “btc” or a currency from this list. (formatted string)
price= the price per unit in the same currency as currency_code. (string)
process_time= the time it will take to prepare the item for shipping (string)
nsfw= is the item nsfw (“true” or “false”)
shipping_origin= where it ships from (required if physical good). must be a formatted string from this list. (formatted string)
shipping_regions= a list of countries/regions where the product will ship to (required if physical good). each item in the list must be formatted from this list. (LIST of formatted strings)
est_delivery_domestic= estimated delivery time for domestic shipments (string)
est_delivery_international= estimated delivery time for international shipments (string)
terms_conditions= any terms or conditions the user wishes to include (string)
returns= return policy (string)
shipping_currency_code= the currency code used to price shipping. may either be “btc” or a currency from this list. (formatted string)
shipping_domestic= the price of domestic shipping (string)
shipping_international= international shipping price
keywords= a list of string search terms. must be fewer than 10. (LIST of strings)
category= a category for this product. will show in store’s category list (string)
condition= the condition of the product (string)
sku= a sku for the item (string)
images= a list of image hashes. the images should be uploaded using the upload_image api call. (LIST of 40 character hex strings)
free_shipping= (“true” or “false”)
moderators= a list of moderator guids the vendor wishes to use (LIST of 40 character hex string guids) Note: the moderator must have been previously returned by the get_moderators websocket call. Given the UI workflow, this call should always be made before the contract is set.
options= a list of options for the product. example: “size”, “color” (LIST of strings)
option= for each option in the options list another argument should be added using that option name and a list of values. for example, given “color” in the options list the next argument should be color= “red”, “white”, “blue” (LIST of strings)

action:

saves the contract to the db and file system. will also publish the keywords in the dht.

return:
{success: True, id: contract_hash}

@DELETE('^/api/v1/contracts')
arguments:
id= the contract id to delete (40 character hex string)
delete_images= include this argument to also delete the images. omit it to keep the images. (“True”)
action:
delete the contract from the file system and db. maybe delete the images as well. sends a delete message to the dht to remove the keywords.
return:
{success: True}

@GET('^/api/v1/shutdown')
arguments:
None
action:
Cleanly disconnects for each node and shuts down the server.
return:
None

@POST('^/api/v1/make_moderator')
arguments:
None
action:
sets the moderator boolean in the profile to True and inserts this node into the dht as a moderator.
return:
{success: True}

@POST('^/api/v1/unmake_moderator')
arguments:
None
action:
sets the moderator boolean in the profile to False and deletes this node from the dht as a moderator.
return:
{success: True}

@POST('^/api/v1/purchase_contract')
arguments:
red is required if physical good

id= the contract id to purchase. Note the contract must be in cache meaning it must have been viewed at least once in the UI. (40 character hex string)
quantity= the quantity purchased (string)
refund_address = the user's refund address (string) (must use correct network, testnet/mainnet)
ship_to= the name of the person to ship to (string)
address= street address (string)
city= city (string)
state= state (string)
postal_code= postal code (string)

moderator= the moderator chosen by the buyer. omit if direct payment (40 character hex string)
options= a list of options for the product. example: "size", "color" (LIST of strings)
option= for each option in the options list another argument should be added using that option name and the buyer's chosen value. for example, given "color" in the options list and a buyer choice of "blue", next argument should be color= "blue"

action:
sends the purchase info to the seller and waits for a response. the user will have 10 minutes to fund the multisig address or direct payment address. If the 10 minutes expires, the purchase order will be deleted and the buyer will have to try again.

response:
```
{
    success: True,
    address: the_bitcoin_address_to_pay,
    amount: the_amount_to_pay,
    order_id: the_order_id
}
```

```
@POST('^/api/v1/confirm_order')
```

arguments:
id= the order id to confirm (40 character hex string)
payout_address= properly formatted bitcoin address where funds should be sent.
comments= any comments to leave for the buyer. can be omitted if none (string)
shipper= the shipper for the item. omit if not physical good (string)
tracking_number= shipment tracking number. omit if not physical good (string)
est_delivery= estimated delivery date. omit if not physical good (string)
url= a download url if digital good. omit if not digital good(string)
password= a password to download a digital good. omit if not digital good (string)

action:
sends the order confirmation and shipping information to the buyer. if he's offline it will stick it in the dht. updates the status of the order in the db.

return:

{success: True}

@POST('^/api/v1/upload_image')
arguments:
image= a list of product images to upload (LIST of images in base64. data only, no base64 prefix)
avatar= use this if uploading an avatar image (base64 image)
header= use this if uploading a header image (base64 image)
action:
saves the image in the file system and a pointer to it in the db
return:
{success: True, image_hashes: [list_of_image_hashes]}

@POST('^/api/v1/complete_order')
arguments:
id= the order id to complete (40 character hex string)
feedback= feedback rating (string “1” - “5”)
quality= quality rating (string “1” - “5”)
description= description rating (string “1” - “5”)
delivery_time= delivery time rating (string “1” - “5”)
customer_service= customer service rating (string “1” - “5”)
review= text review of vendor (string <= 80 characters)
*review fields may be omitted
action:
sends a message containing the payout signature over to the vendor who will release the funds to himself.
return:
{success: True}

@POST('^/api/v1/settings')
all fields are mandatory
arguments:
refund_address= buyer’s refund address (string)
currency_code= may either be “btc” or a currency from this list. (formatted string)
country= the location of the user. must be a formatted string from this list. (formatted string)
language= user’s prefered language (string)
time_zone= the user’s time zone (string)
notifications= display notifications (“True” or “False”)
shipping_addresses= json list of shipping addresses
blocked= a list of guids to block (LIST of 40 character hex strings)
libbitcoin_server= the server address (url string)
ssl= use ssl on the openbazaar server (“True” or “False”)

term_conditions= default terms/conditions (string)
refund_policy= default refund policy (string)
resolver= the blockchain id resolver url (string)
action:
saves the settings into the db
return:
{success: True}

@GET('^/api/v1/settings')
arguments:
None
return:
settings json object

@GET('^/api/v1/get_notifications')
arguments:
limit= how many to return. will start with most recent. defaults to unlimited.
return:
notifications json list

@POST('^/api/v1/mark_notification_as_read')
arguments:
id= the id of the notification (40 character hex string)
action:
marks a notification as read in the database
return:
{success: True}

@POST('^/api/v1/broadcast)
arguments:
message= the message to send (140 characters or less)
action:
sends a broadcast message to all online followers
return:
{success: True, peers reached: num_reached}

@GET('^/api/v1/get_chat_messages')
arguments:
guid= the guid of the node to return the chat messages for
limit= how many to return. will start with most recent. defaults to unlimited.
start= the starting point in the message list
return:
notifications json list

@GET('^/api/v1/get_chat_conversations')
arguments:
    None
return:
    json list containing guid and avatar hash of outstanding conversations.

@DELETE('^/api/v1/chat_conversation')
arguments:
    guid= the guid of the other party (40 character hex string)
action:
    deletes all chat messages with the given guid
return:
    {success: True}

@POST('^/api/v1/mark_chat_message_as_read')
arguments:
    guid= the guid of the other party (40 character hex string)
action:
    marks all messages in a conversation as read in the database
return:
    {success: True}

@GET('^/api/v1/get_sales')
arguments:
    None
return:
    json list containing data for each sale in the database.

@GET('^/api/v1/get_purchases')
arguments:
    None
return:
    json list containing data for each purchase in the database.

@POST('^/api/v1/check_for_payment')
arguments:
    order_id= the id of the order (40 character hex string)
action:
    queries the libbitcoin server to see if any payments were made to the contract's
    funding address. if so it will trigger the websockets to send the payment received
    message as normal.

return:
    {success: True} does not mean a transaction was found. only that the blockchain query was successful.

@GET('^/api/v1/get_order')
arguments:
    order_id= the id of the order (40 character hex string)
return:
    the full json contract in its current state plus the bitcoin transaction info

@POST('^/api/v1/dispute_contract')
arguments:
    order_id= the id of the order (40 character hex string)
    claim= the reason the dispute is being opened (text)
action:
    sends a dispute_open message to both the moderator and other party to the dispute.
return:
    {success: True}

@POST('^/api/v1/close_dispute')
arguments:
    order_id= the id of the order (40 character hex string)
    resolution= the decision of the moderator (text)
    buyer_percentage= percentage the buyer wins
    vendor_percentage= percentage the vendor wins
    moderator_percentage= percentage for moderator fee
    moderator_address= bitcoin address where moderator will receive payment (must be valid)
action:
    signs a payout transaction and sends it to both parties along with the decision
return:
    {success: True}

@POST('^/api/v1/release_funds')
arguments:
    order_id= the id of the order (40 character hex string)
action:
    broadcasts the transaction received from the moderator following a dispute. should only be called if the user approves of the payout distribution.
return:

{success: True}

@GET('^/api/v1/get_cases')
arguments:
None
return:
json list of all the cases in the database

@GET('^/api/v1/get_dispute_messages')
arguments:
order_id= the order id to get the messages for (40 character hex string).
return:
messages json list

@GET('^/api/v1/get_ratings')
arguments:
contract_id= the a contract id to get the ratings for. omit this argument to get the ratings for all contracts. (40 character hex string).
guid= the guid of the node to fetch the ratings from. should be omitted if getting ratings from own node (40 character hex string)
return:
ratings json list