AffectUs UI driven URLs

URL	Actor	Action
http://ip_address:1880/ui/#/0	Product Owner	Enable Access and Declare Dependencies
http://ip_address:1880/ui/#/4	App Dev	Declare Chain
http://ip_address:1880/ui/#/5	Ext Dev	Declare external notification
http://ip_address:1880/ui/#/6	Product Owner	Monitor weather
http://ip_address:1880/ui/#/2	Product Owner	Monitor chain
http://ip_address:1880/ui/#/1	Product Owner	Retrieve chain history

AffectUs Internal API

Protocol	Method or Type	Path	Input	Output
HTTP	POST	/postnotification	{ "eventConditionname": <"stage_change" or "external"> , "statename":<state (e.g. "skip","counterfeit","reverse","delayed") > , "Details":{ "timestamp":<unix time> , "thing_id": <thing id> , "product_id": <product id> , "stage": <Chain Stage> } , <extra fields to be ralayed> }	{ "input": <original_input> , "message":"Event received...", "endTime": <unix time> }
HTTP	POST	/stageCorrelation	{ "timestamp": <unix time>, "location": <valid GeoJSON>, "product_id": <product id>, "thing_id": <thing id>, <extra fields to be relayed> }	{ "message" : "Scan event received, processing scan..." }
WebSocket	Listener	/ws/liveFauxStream	-	{ "timestamp": <unix time>, "location": <Randomized valid GeoJSON>, "stage": <DEFAULT Stage>, "product_id": <Randomized valid product ID>, "thing_id": <Randomized valid thing ID> }
MongoDB Wire	2-way TCP/IP Socket	mongodb://ip_of_MongoDB:27017/affectus	<MongoDB record in JSON format>	<MongoDB response>

HTTP	POST	http://ip_of_AI_service:5000	{ 'data': <JSON Array of feature tupples>, 'parameters':{ 'retrain': <True or False>, 'refresh_data': <True or False>, 'train_file': <train data path>, 'model_dir': <path to store the model>, 'model_parameters': <Other model parameters in JSON format> } }	{'error': <errors in JSON format>, 'data_refresh_output': <status and errors of data refresh>, 'retrain_output': <status and errors of retraining>, 'predictions': { 'predictions': <JSON Array of predicted durations>, 'actuals': <JSON array of actual durations if available>, 'costs': <JSON array of prediction errors if actual is available> }}
HTTP	POST	http://localhost:1880/durationRequest	[{ 'thing_id': <thing id>, 'product_id': <product id>, 'data':[{ 'stage': <integer stage index>, 'day': <day of the week>, 'hour': <hour of the day>, <other prediction features> }, {<features JSON 2>}, …, {<features JSON N>}] }, {<thing 2 entry>}, …, {<thing K entry>}]	[{ 'thing_id': <thing id>, 'expected_durations' : <prediction model results> }, {<thing 2 predictions>}, …, {<thing K predictions>}]
HTTP	GET	http://localhost:1880/weather	• lat = <latitude as string> • lon = <longitude as string> • window = <window size as integer>	{ "events": <JSON array of weather related events>, "response": { "temperature": <-1, 0 or 1>, "wind": <0 or 1>, "humidity": <-1, 0 or 1>}, "readings": { "temperature": <temperature reading>, "wind": <wind reading>, "humidity": <humidity reading>}, "fullData": <full data returned by external API> }

AffectUs Used API from other frameworks

Framewor	Protocol	Method	Path	Description	reference

k					
EVT	HTTP	GET	/products	Read all products	https://developers.evrythng.com/docs/api-key-scopes-and-permissions
EVT	HTTP	GET	/thngs	Read all things	
EVT	HTTP	POST	/thngs/<thingId>/properties	Create/update properties of a thing	
EVT	MQTT	Subscribe	mqtts://mqtt.evrythng.com:8883/actions	Action type is created	https://developers.evrythng.com/docs/pubsub#section-mqtt
EVT	MQTT	Publish	mqtts://mqtt.evrythng.com:8883/actions	Create an action type	
Docker	Direct client commands (CLI)	Pull	https://hub.docker.com/u/affectus/	Download (Pull) locally an image from the affectus hub	https://docs.docker.com/engine/reference/commandline/cli/#option-types
		Create		Create a container from the downloaded image	
		Run		Deploy (Run) the created container	
		Tag		(re)Tag the containing image of the running container	
		Push		Push the new image to the affectus account on the Docker Hub	
Fuseki	SPARQL over HTTP	“Select” or s-query	http://<Fuseki_IP>:3030/**affectus**/sparql	Serves the **knowledge base**. All the existing and inferred triples are available at /sparql, while updating the triples is supported at /update	https://jena.apache.org/documentation/fuseki2/soh.html
		“Update” or s-update	http://<Fuseki_IP>:3030/**affectus**/update		
		“Select” or s-query	http://<Fuseki_IP>:3030/**ontology**/sparql	Serves the **ontology**, as a static, queryable OWL graph (Terminological box)	
RabbitMQ	AMQP	Publish	Depends on user declaration, if they provide a path, the respective notification is forwarded to their messaging system		https://www.rabbitmq.com/tutorials/tutorial-one-javascript.html
-	HTTP	Post	Depends on user declaration, if they		-

			provide a path, the respective notification is forwarded to their service endpoint	