PIVIOT

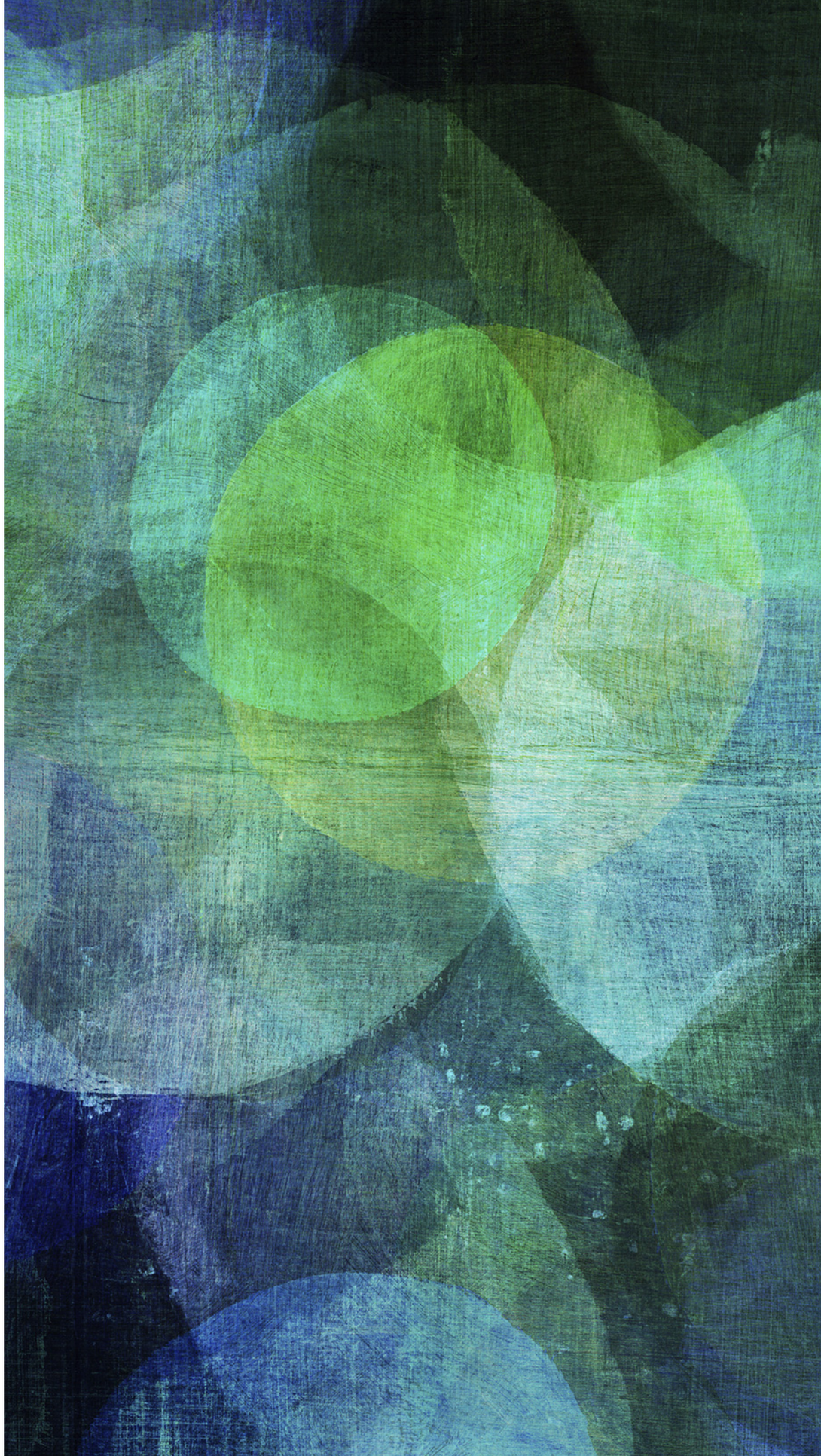
An platform for gaining remote control over applications. … ?

MEETING OUTCOME GOALS:

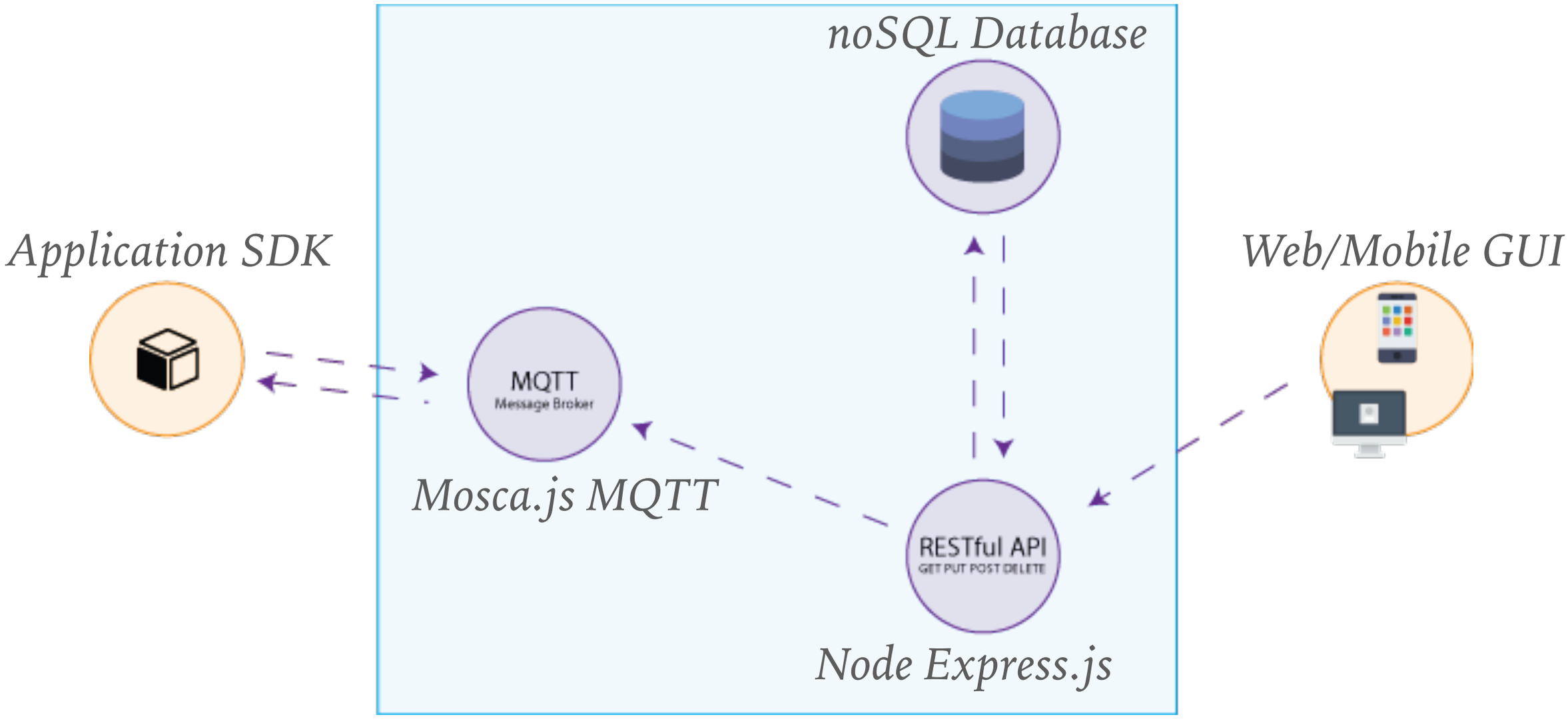
- ~25 min - Presentation of proposed capstone project.
  - Outcome: **Approval** or Disapproval of proposal
    - ~ 5 min - If not approved present 3 alternative ideas and approve.
      - Mobile App for a Conifer (Trees) database.
      - Build an application for a local art museum.
        - Description: Be able to answer question through an app while guests are walking through the museum.
      - Rebuild local museums website?

SYSTEM ARCHITECTURE

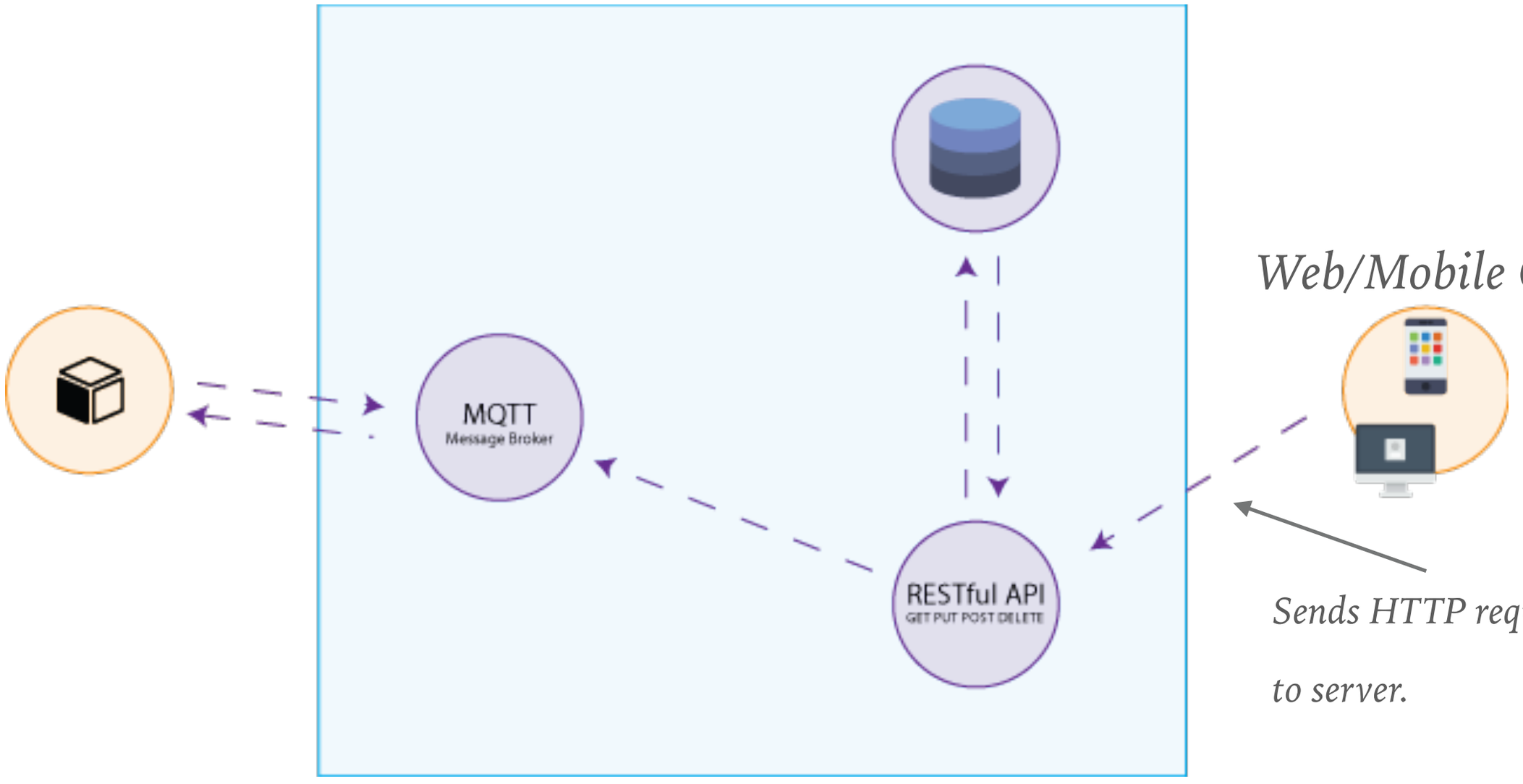
An Overview



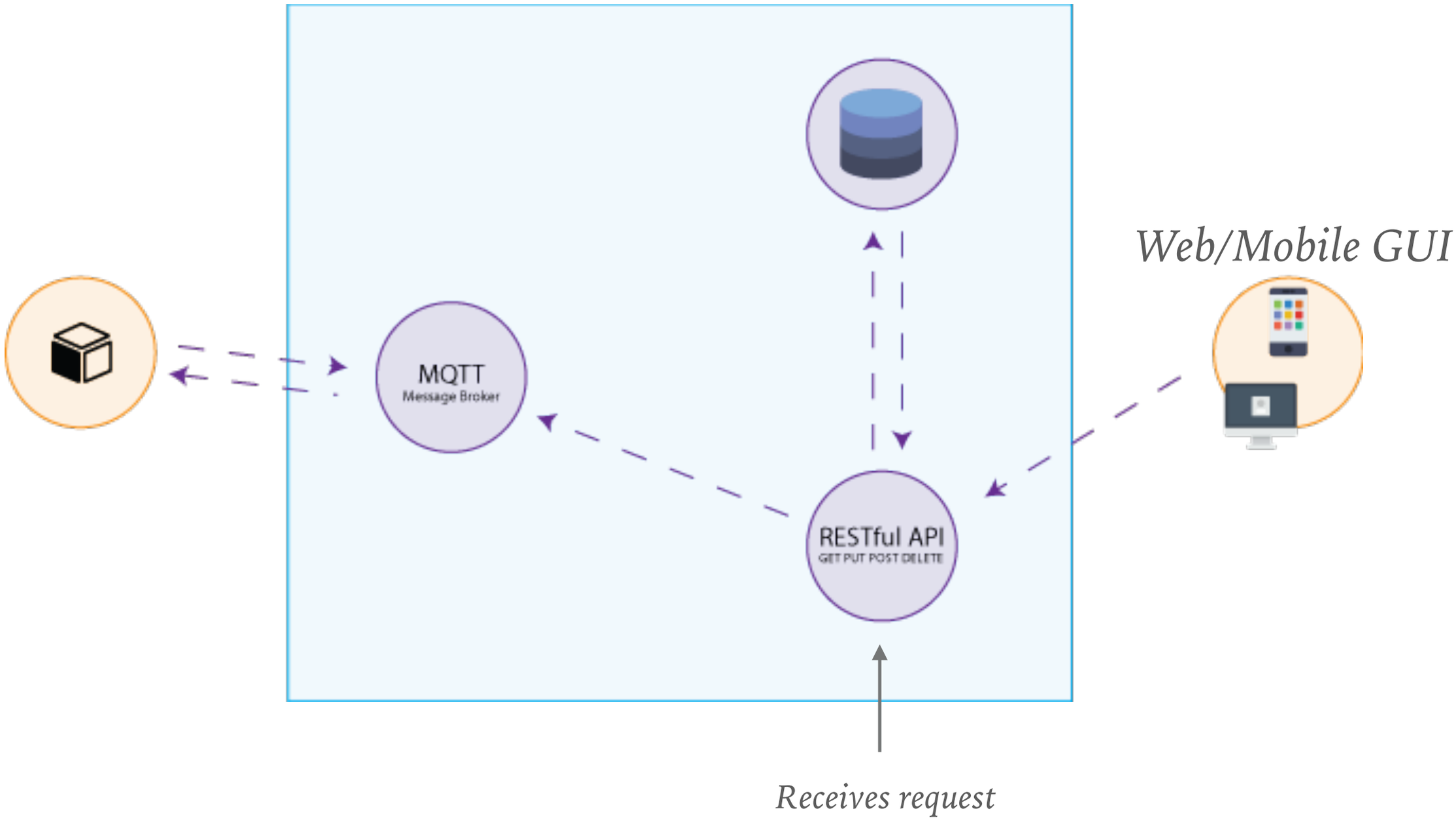
SYSTEM ARCHITECTURE



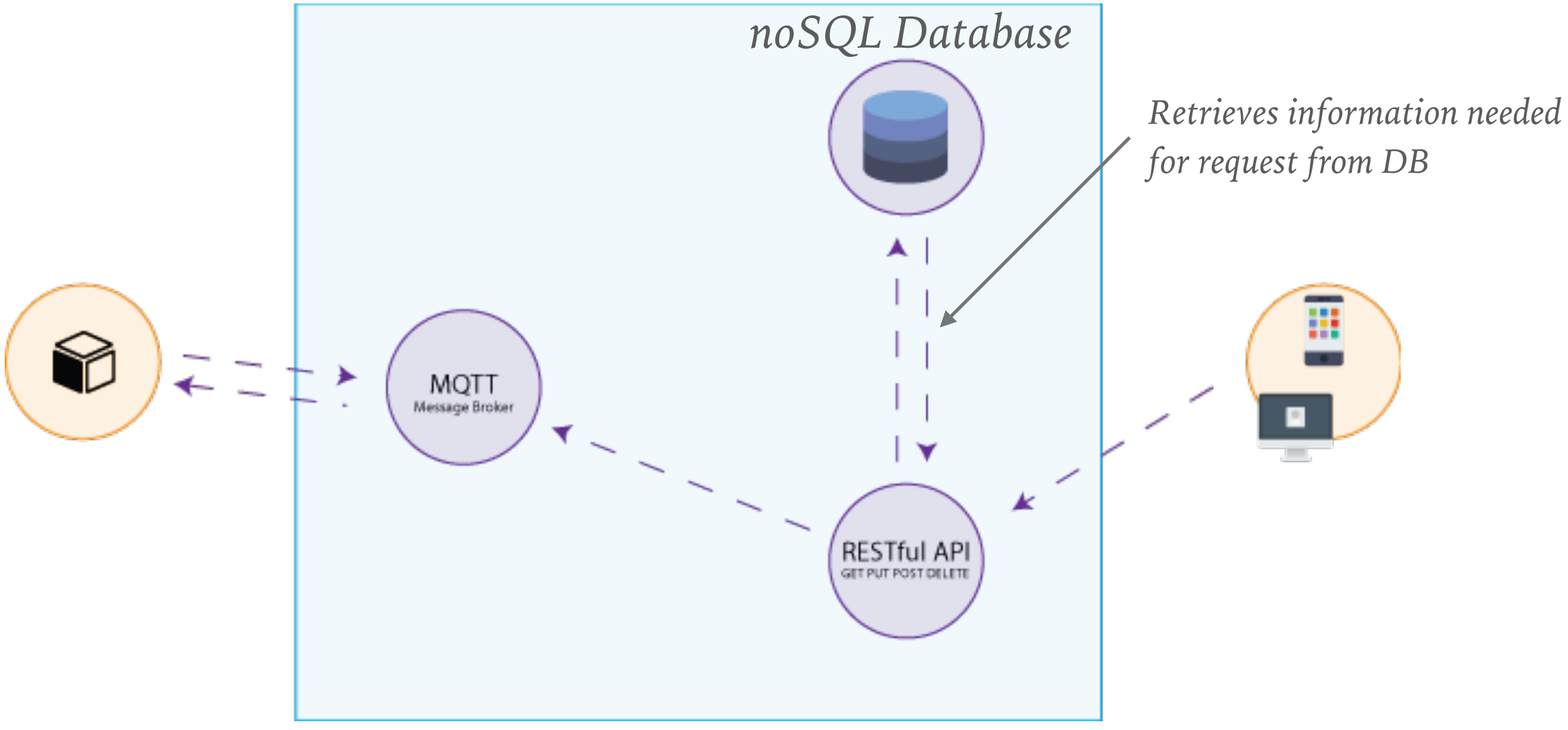
SYSTEM ARCHITECTURE



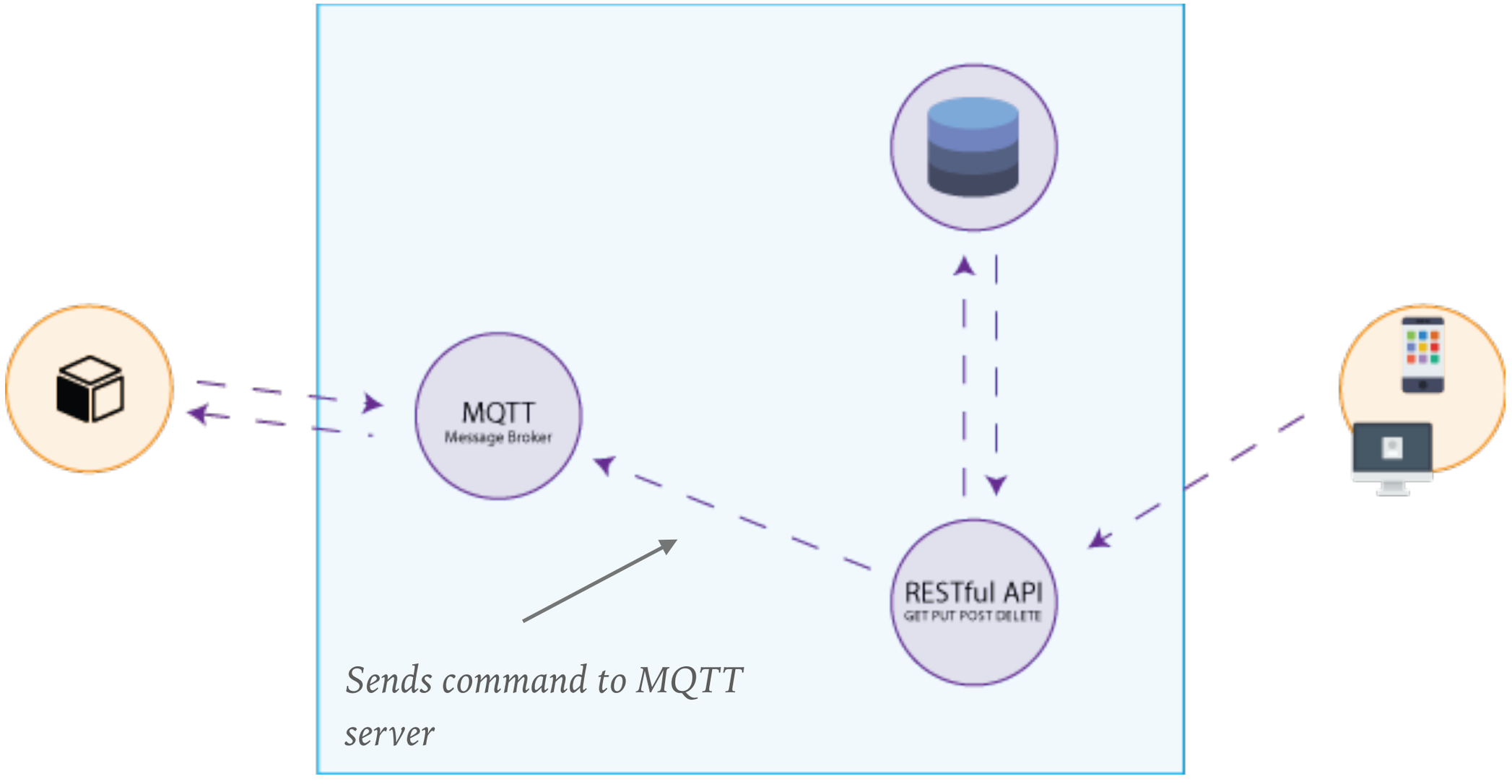
SYSTEM ARCHITECTURE



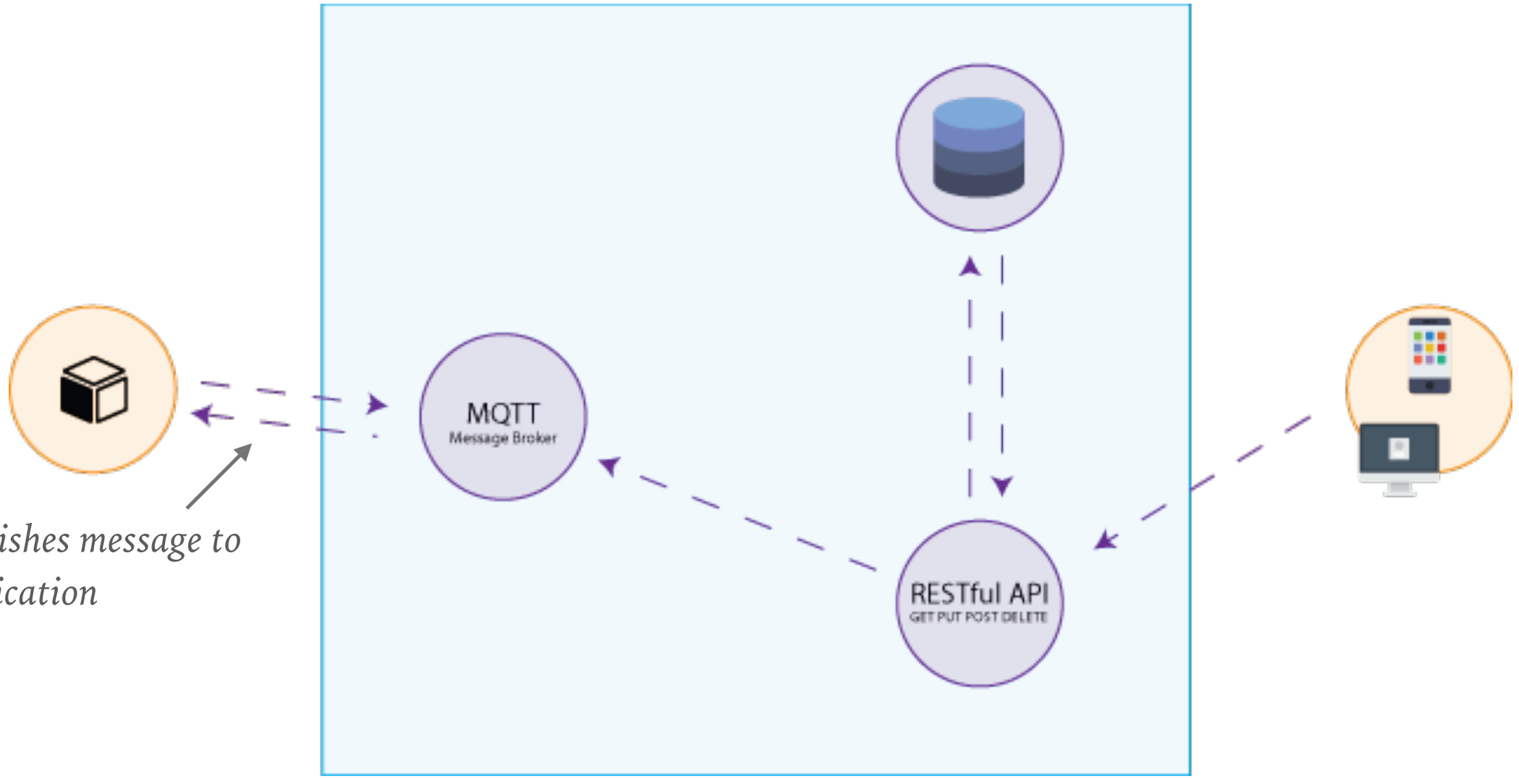
SYSTEM ARCHITECTURE



SYSTEM ARCHITECTURE

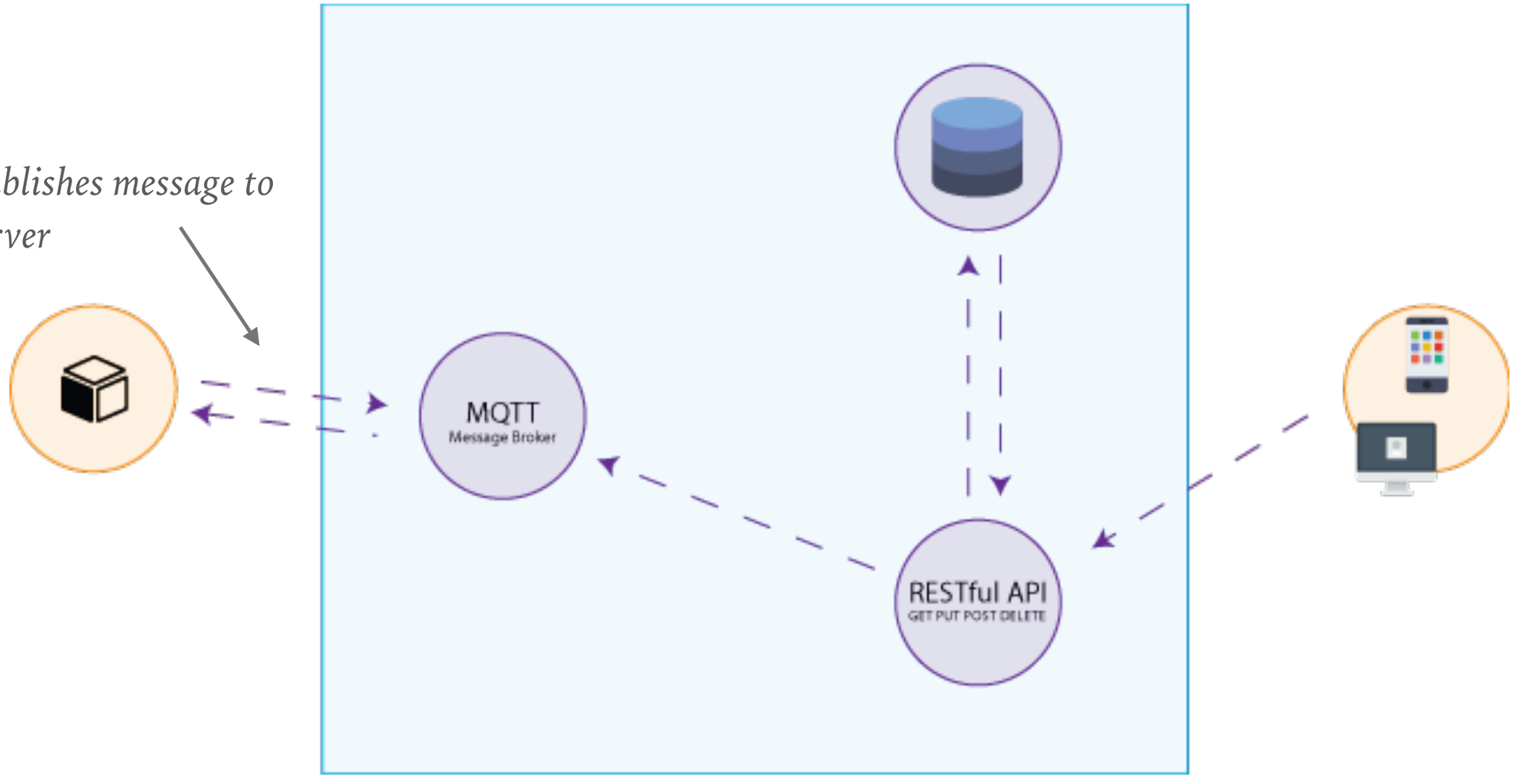


SYSTEM ARCHITECTURE

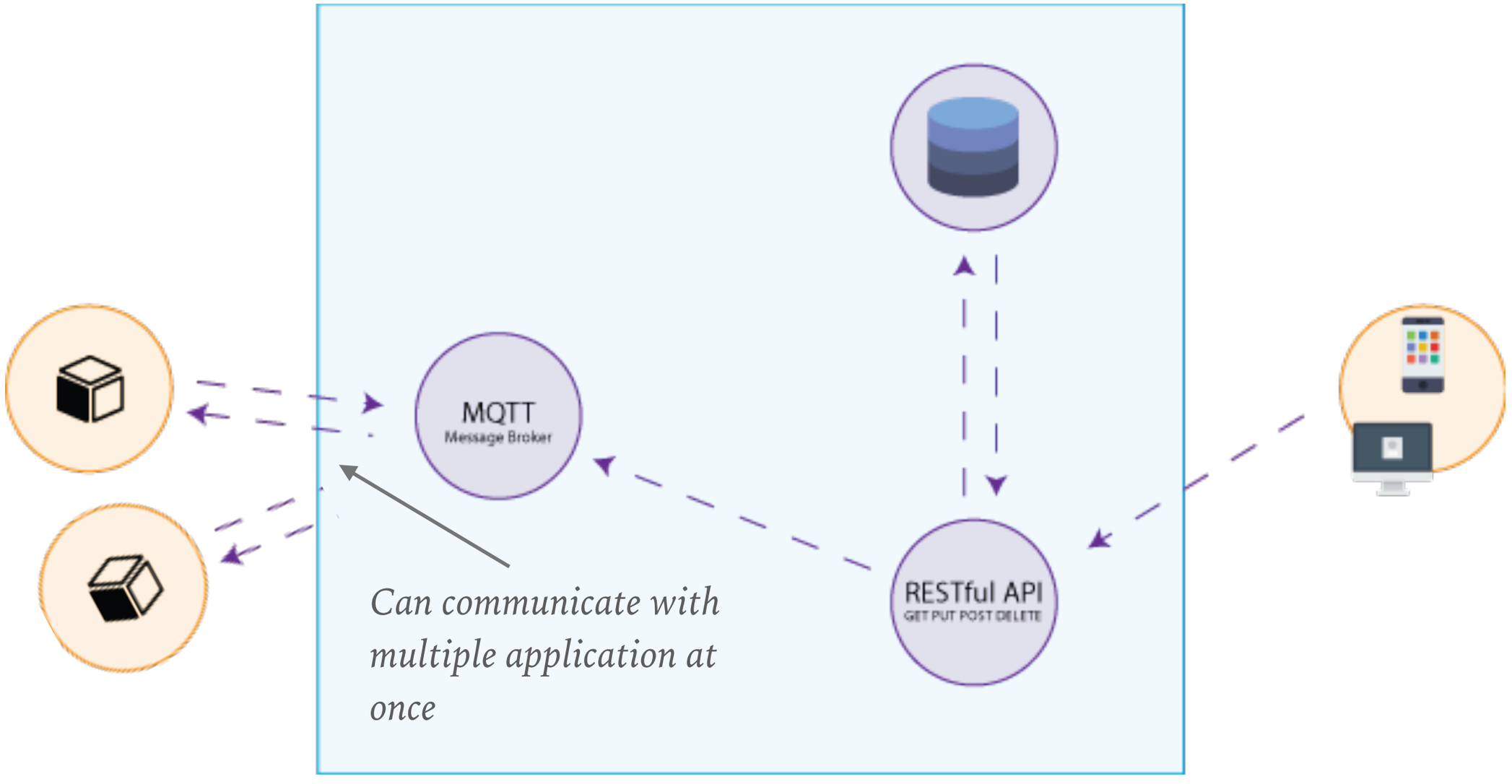


Publishes message to application

SYSTEM ARCHITECTURE

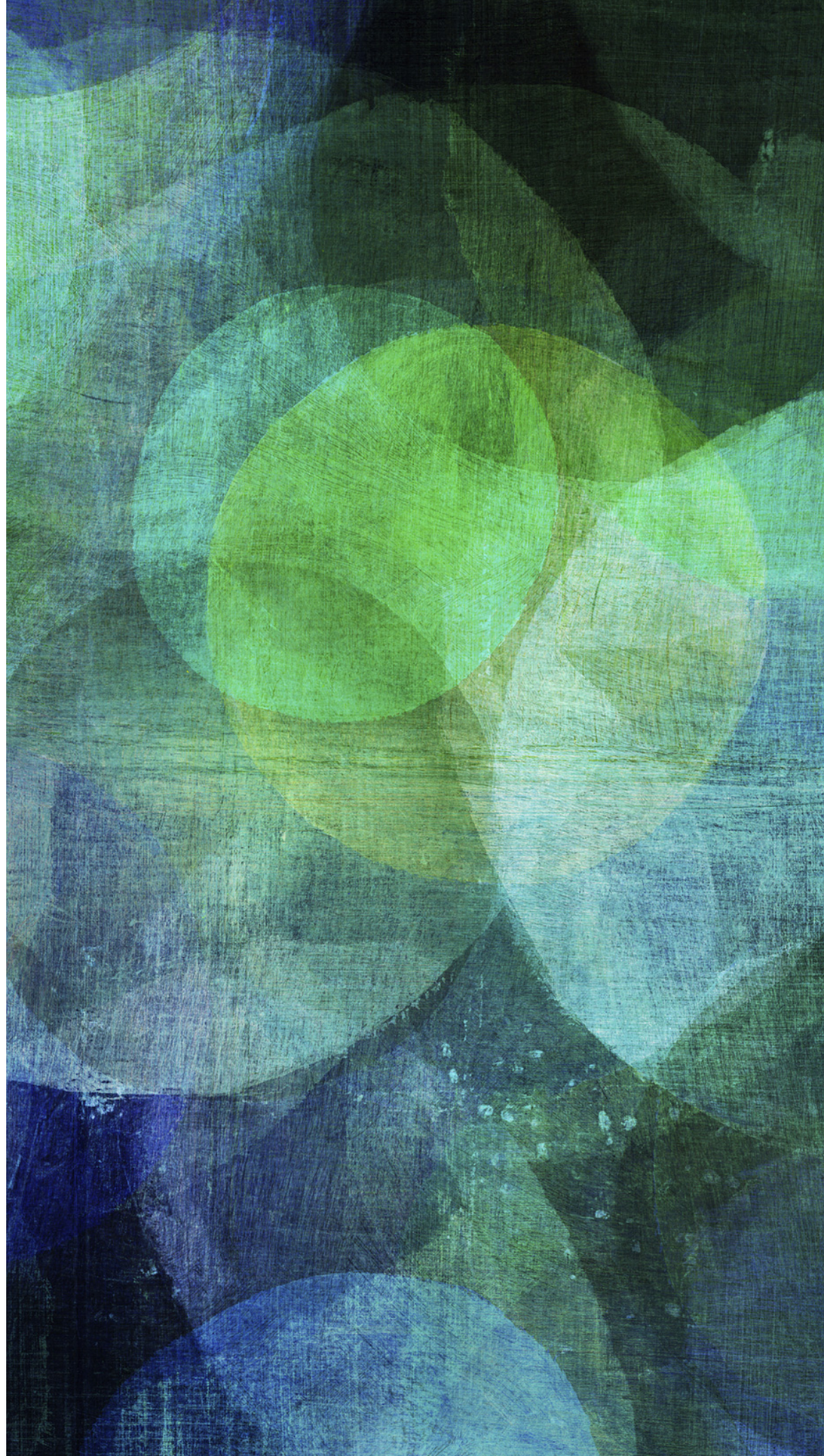


SYSTEM ARCHITECTURE



FROM THE SERVER SIDE

A look at the server side implementation



NODEMON AP-SERVER.JS —— STARTING THE SERVER

```
// GLOBAL VARIABLES
require('./globals.js')
```

Global Variables

NODEMON AP-SERVER.JS —— STARTING THE SERVER

Global Variables

```
settings = require('./config/settings.js')
```

NODEMON AP-SERVER.JS —— STARTING THE SERVER

Settings.js

```
module.exports = {
      RESTful:{
            port: 3000
      },
      crypto:{
            secret: "%0Bb-$sCJ+u2rq6r'3|sg#)j1?[dj/,82*%xXkH~QheJ2NvN%,W.\"Bal29DvLJP"
      },
      mongo: {
     uri: 'mongodb://capstone-admin:zuRM3NXycfQN4FZRbBaAQEAXSw7RLn8F5HfRL8tDBXThDTk6vwH8@ds063946.mlab.com:63946/capstone',
            hostname: 'zaphyrr.com',
            db: 'main',
            port: 27017
      }
}
```

This file is just used to store settings of the application so that we may share it across the application.

NODEMON AP-SERVER.JS —— STARTING THE SERVER

Global Variables

```
settings = require(‘./config/settings.js')


//VARIOUS LIBRARIES
jwt = require('jwt-simple');
Promise = require('promise');
crypto = require('crypto');
bcrypt = require('bcrypt');
bodyParser  = require('body-parser');
morgan = require('morgan');
randtoken = require('rand-token');
cookieParser = require('cookie-parser');
NodeRSA = require('node-rsa');
passport = require('passport');
cors = require('cors');
HashMap = require('hashmap');
async = require("async");
```

NODEMON AP-SERVER.JS —— STARTING THE SERVER

Global Variables

```
settings = require(‘./config/settings.js')


//VARIOUS LIBRARIES
jwt = require('jwt-simple');
Promise = require('promise');
crypto = require('crypto');
bcrypt = require('bcrypt');
bodyParser  = require('body-parser');
morgan = require('morgan');
randtoken = require('rand-token');
cookieParser = require('cookie-parser');
NodeRSA = require('node-rsa');
passport = require('passport');
cors = require('cors');
HashMap = require('hashmap');
async = require("async");



//Some Data Structures
ArrayList = require('./classes/ArrayList.js');
Queue = require(‘./classes/Queue.js'); //for queues
Graph = require(‘node-dijkstra') //for graphing
fs = require(“fs") //file reader



// RESTFUL SERVICES
rest = require('express')();
RESTfulServices = require('http').Server(rest);
rest.set('superSecret', settings.crypto.secret);
```

NODEMON AP-SERVER.JS —— STARTING THE SERVER

Global Variables

```
settings = require(‘./config/settings.js')


//VARIOUS LIBRARIES
jwt = require('jwt-simple');
Promise = require('promise');
crypto = require('crypto');
bcrypt = require('bcrypt');
bodyParser  = require('body-parser');
morgan = require('morgan');
randtoken = require('rand-token');
cookieParser = require('cookie-parser');
NodeRSA = require('node-rsa');
passport = require('passport');
cors = require('cors');
HashMap = require('hashmap');
async = require("async");


//Some Data Structures
ArrayList = require('./classes/ArrayList.js');
Queue = require(‘./classes/Queue.js'); //for queues
Graph = require(‘node-dijkstra') //for graphing
fs = require(“fs") //file reader


// RESTFUL SERVICES
rest = require('express')();
RESTfulServices = require('http').Server(rest);
rest.set('superSecret', settings.crypto.secret);
```

```
//bodyparser (authentication ignore for now)
rest.use(bodyParser.urlencoded({extended:true}));
rest.use(bodyParser.json());
rest.use(cookieParser());
rest.use(morgan('dev'));
rest.use(cors());
rest.use(passport.initialize());

//passport stuff
passport.serializeUser(function(user,done){
      done(null, user._id);
})

passport.deserializeUser(function(id, done){
      User.findById(id, function(err, user){
            done(err,user)
      })
})



// Database Connection
Mongoose = require('./classes/Mongoose.js')
mongoose = new Mongoose(settings)

//Automata Processor
Automaton = require('./classes/Automaton.js');

// MODELS
User = require('./classes/User.js')
ThingSchema = require('./classes/Thing.js').schema
Thing = require('./classes/Thing.js').class
```

NODEMON AP-SERVER.JS —— STARTING THE SERVER

```
// GLOBAL VARIABLES
require('./globals.js')

// START THE MQTT SERVER
Mosca = require('./classes/Mosca.js');
mosca = new Mosca(settings)
```

A FEW NOTES ON NODE.JS MODULES

- JavaScript by default can't "include" other scripts.
- Using Node's require method allows for this.
- The code can `require` modules
  - Allow for 'requiring' dependencies

Credit: https://darrenderidder.github.io/talks/ModulePatterns/#/2

SIMPLE MODULE EXAMPLES

```
//EXAMPLE 1
// magic.js
console.log(“magic!”)

//app.js
require(“magic.js”)
```

SIMPLE MODULE EXAMPLES

```
//EXAMPLE 1
// magic.js
console.log(“magic!”)

//app.js
require(“magic.js”)
```

```
//EXAMPLE 2 Declaring Global
//magic_global.js
magic = function(){
   console.log(“magic!”)
}

//app.js
require(“magic.js”)
magic()
```

SIMPLE MODULE EXAMPLES

```
//EXAMPLE 1
// magic.js
console.log(“magic!”)

//app.js
require(“magic.js”)
```

```
//EXAMPLE 2 Declaring Global
//magic_global.js
magic = function(){
   console.log(“magic!”)
}

//app.js
require(“magic.js”)
magic()
```

```
//EXAMPLE 3 Exporting Module stuff
//magic_global.js
module.exports = function(){
   console.log(“magic!”)
}

//app.js
var magic = require(“magic.js”)
magic()
```

SIMPLE MODULE EXAMPLES

```
//EXAMPLE 1
// magic.js
console.log(“magic!”)

//app.js
require(“magic.js”)
```

```
//EXAMPLE 2 Declaring Global
//magic_global.js
magic = function(){
   console.log(“magic!”)
}

//app.js
require(“magic.js”)
magic()
```

```
//EXAMPLE 3 Exporting Module stuff
//magic_global.js
module.exports = function(){
   console.log(“magic!”)
}

//app.js
var magic = require(“magic.js”)
magic()
```

Cool…

SIMPLE MODULE EXAMPLES

Cool… But Even Cooler … and how I am using it in my application

Javascript Function

```
function Foo(){
  //do some cool constructor stuff
}

Foo.prototype = {
  attribute1: ‘bar’,
  function: function(){
    console.log(“Hello”)
  }
}

module.exports = Foo
```

```
//app.js
var Foo = require(‘Foo.js’)

var foo = new Foo();
console.log(foo.attribute1)
```

SIMPLE MODULE EXAMPLES

Cool… But Even Cooler … and how I am using it in my application

Javascript Function

```
function Foo(){
  //do some cool constructor stuff
}
```

Javascript Prototype

```
Foo.prototype = {
  attribute1: ‘bar’,
  function: function(){
    console.log(“Hello”)
  }
}

module.exports = Foo
```

```
//app.js
var Foo = require(‘Foo.js’)

var foo = new Foo();
console.log(foo.attribute1)
```

SIMPLE MODULE EXAMPLES

Cool… But Even Cooler … and how I am using it in my application

Javascript Function

```
function Foo(){
  //do some cool constructor stuff
}
```

Javascript Prototype

```
Foo.prototype = {
  attribute1: ‘bar’,
  function: function(){
    console.log(“Hello”)
  }
}
```

Node.js Exporting

```
module.exports = Foo
```

```
//app.js
var Foo = require(‘Foo.js’)

var foo = new Foo();
console.log(foo.attribute1)
```

SIMPLE MODULE EXAMPLES

Cool… But Even Cooler … and how I am using it in my application

```
function Foo(){
  //do some cool constructor stuff
}
```

Javascript Function

```
Foo.prototype = {
  attribute1: ‘bar’,
  function: function(){
    console.log(“Hello”)
  }
}
```

Javascript Prototype

```
module.exports = Foo
```

Node.js Exporting

```
//app.js
var Foo = require(‘Foo.js’)

var foo = new Foo();
console.log(foo.attribute1)
```

Oh Look it’s basically a class

SIMPLE MODULE EXAMPLES

Cool… But Even Cooler … and how I am using it in my application

```
function Foo(){
  //do some cool constructor stuff
}
```

Javascript Function

```
Foo.prototype = {
  attribute1: ‘bar’,
  function: function(){
    console.log(“Hello”)
  }
}
```

Javascript Prototype

```
module.exports = Foo
```

Node.js Exporting

```
//app.js
var Foo = require(‘Foo.js’)

var foo = new Foo();
console.log(foo.attribute1)
```

Oh Look it’s basically a class

And now it’s an object.

Object-Oriented Programming using Node.js

NODEMON AP-SERVER.JS —— STARTING THE SERVER

```
// GLOBAL VARIABLES
require('./globals.js')

// START THE MQTT SERVER
Mosca = require('./classes/Mosca.js');
mosca = new Mosca(settings)
```

NODEMON AP-SERVER.JS —— STARTING THE SERVER

Mosca Class (Mosca.js)

```
var mosca = require('mosca');


//CLASS TO START THE MQTT SERVER
function Mosca(){

      this.ascoltatore = {
        //using ascoltatore
        type: 'mongo',
        url: settings.mongo.uri,
        pubsubCollection: 'MQTT',
        mongo: {}
      },

      this.settings = {
            port: 1883,
            backend: this.ascoltatore,
            http: {
          port: 1884,
          bundle: true,
          static: './'
        }
      },

      //Create the server
      this.server = new mosca.Server(this.settings)
      //get some listener functions
      this.server.on('clientConnected', this.clientConnected);
      this.server.on('clientDisconnecting', this.clientDisconnecting);
      this.server.on('clientDisconnected', this.clientDisconnected);
      this.server.on('published', this.published);
      this.server.on('delivered', this.delivered);
      this.server.on('subscribed', this.subscribed);
      this.server.on('unsubscribed', this.unsubscribed);
      // fired when a message is received
      //when the server is ready let us know please
      this.server.on('ready', this.setup);
}
```

```
Mosca.prototype = {
      setup: function(){
            console.log('MQTT up');
      },

      clientConnected: function(client){
            console.log('connected', client.id)
      },

      clientDisconnecting: function(client){
            console.log('disconnecting', client.id)
      },

      clientDisconnected: function(client){
            console.log('disconnected', client.id)
      },

      published: function(packet, client){
            console.log('published', packet)
            console.log('from', client)
      },

      delivered: function(packet, client){
            console.log('delivered', packet)
            console.log('to', client)
      },

      subscribed: function(topic, client){
            console.log(client, 'subscribed')
            console.log('to', topic)
      },

      unsubscribed: function(topic, client){
            console.log(client, 'unsubscribed')
            console.log('from', topic)
      }

}
```

module.exports = Mosca

NODEMON AP-SERVER.JS —— STARTING THE SERVER

```
// GLOBAL VARIABLES
require('./globals.js')

// START THE MQTT SERVER
Mosca = require('./classes/Mosca.js');
mosca = new Mosca(settings)



 /**
  *
  *
  * RESTful ROUTES
  *
  *
  */

rest.get('/api/v1', function(req, res){
      res.send("RESTful Services")

})

//PLACE WHERE AUTHENTICATION WOULD BE IMPLEMENTED
// rest.get('/api/v1/authorize', function(req, res){
// })
// rest.get('/api/v1/device/authorize', function(req,res){
// })

//USERS
rest.get('/api/v1/users', function(req, res) {
  User.find({}, function(err, users) {
    res.json(users);
  });
});
```

Continued…

NODEMON AP-SERVER.JS —— STARTING THE SERVER

```
//DEVICE ROUTES
rest.route('/api/v1/devices')
     .get(function(req, res){
     })
     .post(function(req, res){
     })
//SINGLE DEVICE ROUTES
rest.route('/api/v1/device/:id')//match all routes
     .get(function(req, res){        // GET THE REQUESTED DEVICE
          ap = new Automaton(req.params.id); //ignore this name please
          ap.M.findOne({_id: req.params.id}, function(err, model){
               if(err) handleError(err);
               res.send(model)
          })
     })
     .put(function(req, res){        // UPDATE WHOLE DEVICE MODEL
     })
     .patch(function(req,res){       // UPDATE ONLY CHANGED ATTRIBUTES (just implement the same)
     })
     .delete(function(req, res){     // DELETE THE SPECIFIED DEVICE
     })

// Trigger an action
rest.route('/api/v1/device/:id/action/:action')
     .get(function(req, res){
          mosca.server.publish({
               topic: req.params.id + '/transition/' + req.params.action,
               payload: 'Message'
          }, function(){
               res.send("Pushed")
          })
     })
//Turn the restful service on
RESTfulServices.listen(settings.RESTful.port, function(){
     console.log('RESTful port 3000')
})
```

2 things

NODEMON AP-SERVER.JS —— STARTING THE SERVER

```
rest.route('/api/v1/device/:id')//match all routes
     .get(function(req, res){        // GET THE REQUESTED DEVICE
          ap = new Automaton(req.params.id); //ignore this name please
          ap.M.findOne({_id: req.params.id}, function(err, model){
               if(err) handleError(err);
               res.send(model)
          })
     })
```

1. *When a client visits api/v1/device/:id (:id is the device id)*
2. *If the request type is a GET request passion the request and response information to a function*
3. *Create a new "Automaton" graph.*

AUTOMATON.JS CLASS

```
function Automaton(M){
      //Temp Values
      this.machine = null
      this.nodes = null
      this.node_graph = null
      this.paths = null

      //Add the model..
      if(typeof(M) == "string"){
            //EXISTING MODEL
            this._id = M
            var promise = this.fetch(this._id)
      }
      else{
            //NEW MODEL
            this.machine = M
            this.nodes = []
            this.node_graph = {}
            for(var k in this.machine.states){
                  if(this.node_graph[k] == undefined){
                        this.node_graph[k] = {}
                  }
                  for(var l in this.machine.states[k].transitions){
                        this.node_graph[k][l] = 1
                  }

                  this.nodes.push(k)
            }
            this.route = new Graph(this.node_graph)
            this.computePaths()
            this.model = new this.M({
                  machine: this.machine,
                  nodes: this.nodes,
                  node_graph: this.node_graph,
                  paths: this.paths
      })
      this.save()
      }
}
```

AUTOMATON.JS CLASS – PROTOTYPE

```
Automaton.prototype = {
      M: mongoose.mg.model('Automaton', mongoose.mg.Schema({
            machine: mongoose.mg.Schema.Types.Mixed,
            nodes: [mongoose.mg.Schema.Types.Mixed],
            node_graph: mongoose.mg.Schema.Types.Mixed,
            paths: mongoose.mg.Schema.Types.Mixed
      }, {
        toObject: {
        virtuals: true
        },
        toJSON: {
        virtuals: true
        }
      })),
```

DB Schema for Mongoose

AUTOMATON.JS CLASS - PROTOTYPE

```
computePaths: function(){
            var paths = {}
            var count = 0
            for(var i = 0; i < this.nodes.length; i++){
                  for(var j = 0; j < this.nodes.length; j++){
                        // get the path from i to j
                        var path = this.route.path(this.nodes[i], this.nodes[j]);
                        // the path will be null if it is trying to get a path to itself. i.e. 1->1
                        if(path != null){
                              path_label = this.nodes[i] + '-' + this.nodes[j]
                              // create a null Q variable
                              var Q = null
                              // console.log("i = " + i + "j = " + j + "k = " + k)
                              for(var k = 0; k < path.length - 1; k++){
                                    // create the label
                                    label = path[k] + '-' + path[k+1]
                                    fname = this.machine.states[path[k]].transitions[path[k+1]].fname
                                    dependency = this.machine.states[path[k]].transitions[path[k+1]].dependency
                                    // THIS IF STATEMENT SHOULD BE HANDLED IN THE QUEUE CLASS.
                                    if(Q == null){
                                          if(dependency != undefined){
                                                Q = new Queue('dep', dependency)
                                                Q.enqueue(label, fname)
                                          }
                                          else{
                                                Q = new Queue(label, fname)
                                          }
                                    }
                                    else{
                                          if(dependency != undefined){
                                                Q.enqueue('dep', dependency)
                                          }
                                          Q.enqueue(label, fname)
                                    }
                              }
                              paths[path_label] = Q.toArray()
                              this.paths = paths
                        }
                  }
```

AUTOMATON.JS CLASS – PROTOTYPE

```
getPaths: function(){
      return this.paths;
},

getPath: function(from, to){
      var label = from + '-' + to
      return this.paths[label]
},

fetch: function(id){
      var MOD = null
      this.M.findOne({_id: id}, function(err, model){
            if(err) handleError(err);
            MOD = model
                  console.log(MOD)
            })
            console.log(MOD)
      },

      save: function(){

      }
}

module.exports = Automaton ;
```

Just Return Paths

Return just a single path

Fetch from the DB

Mongoose Function

NODEMON AP-SERVER.JS —— STARTING THE SERVER

```
rest.route('/api/v1/device/:id')//match all routes
     .get(function(req, res){        // GET THE REQUESTED DEVICE
          ap = new Automaton(req.params.id); //ignore this name please
          ap.M.findOne({_id: req.params.id}, function(err, model){
               if(err) handleError(err);
               res.send(model)
          })
     })
```

1. *When a client visits api/v1/device/:id (:id is the device id)*
2. *If the request type is a GET request passion the request and response information to a function*
3. *Create a new "Automaton" graph.*
4. *The next function is so send the DB entry back to the client*

NODEMON AP-SERVER.JS —— STARTING THE SERVER

```
rest.route('/api/v1/device/:id/action/:action')
     .get(function(req, res){
          mosca.server.publish({
               topic: req.params.id + '/transition/' + req.params.action,
               payload: 'Message'
          }, function(){
               res.send("Pushed")
          })
     })
```

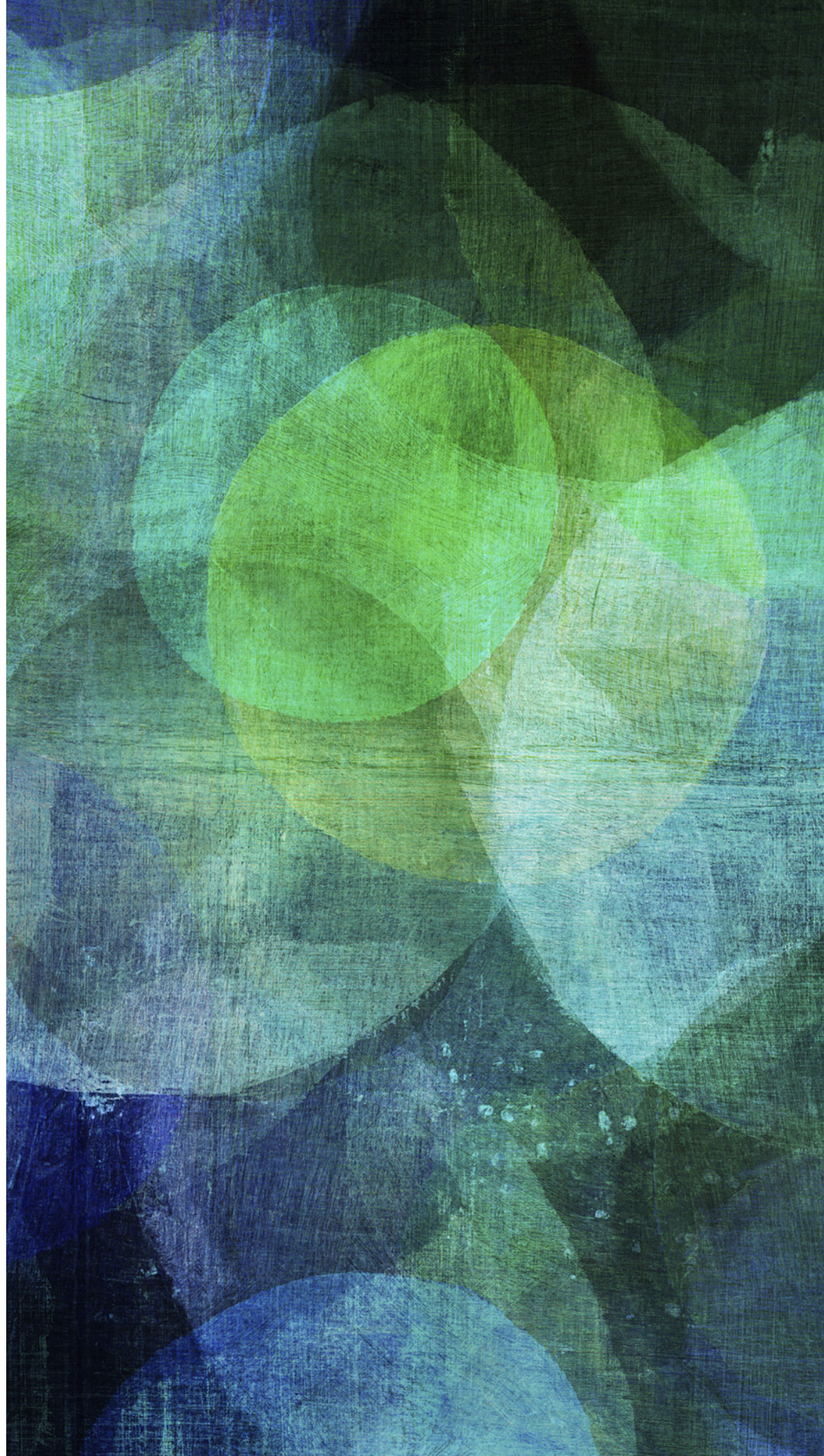
1. *When a client visits api/v1/device/:id/action/:action*
    1. *:id is the device/application id*
    2. *:action is the name of the function to call (explained in a bit)*
2. *If the request type is a GET request passion the request and response information to a function*
3. *Publish a message to the MQTT server with the topic to post to*
4. *Send acknowledgement of sent message to client*

NODEMON AP-SERVER.JS —— STARTING THE SERVER

Good place to switch gears

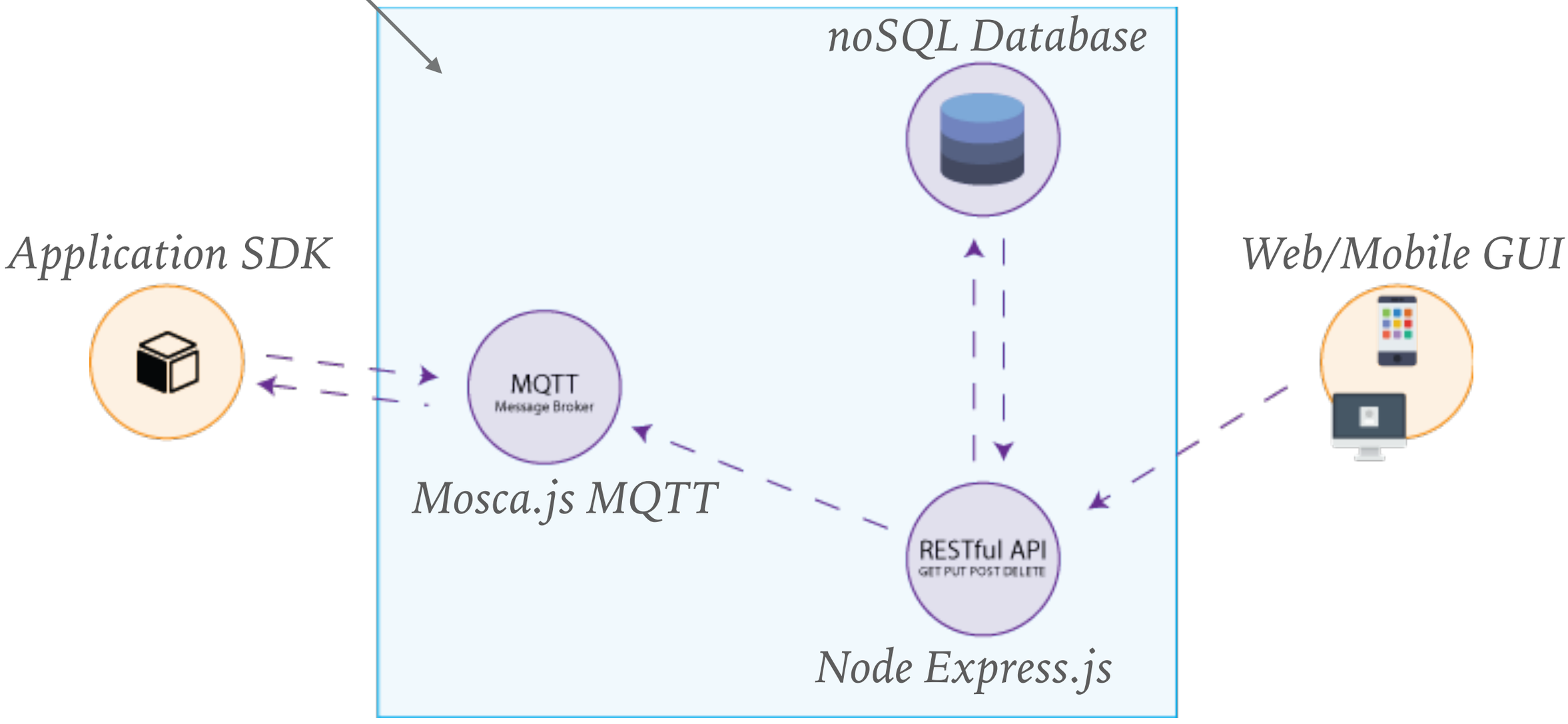
FROM THE USER SIDE

A look at the user implementation

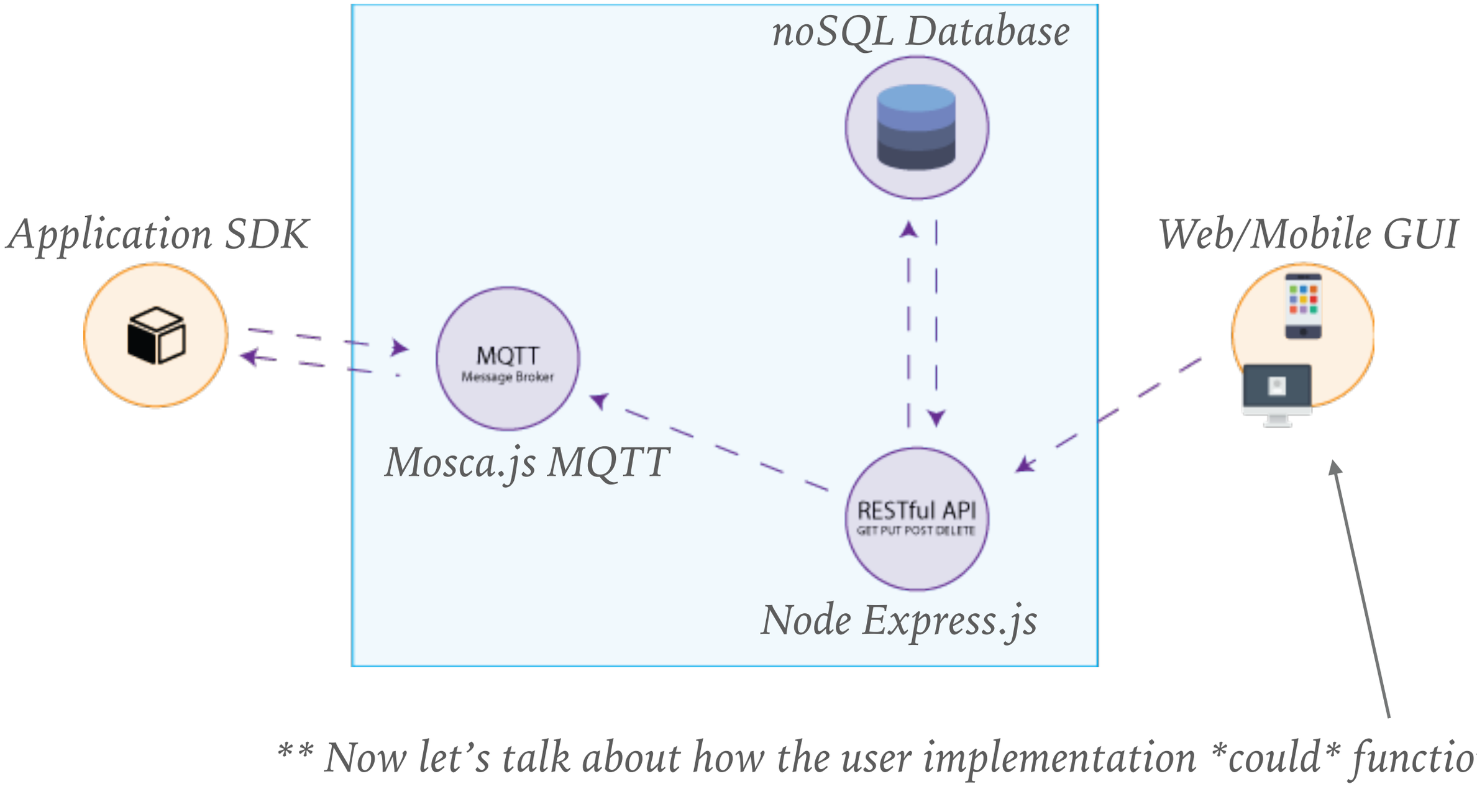


RECAP

We discussed the inner working of the server



RECAP



** Now let's talk about how the user implementation *could* function

SIMULATION OF PLATFORM

- MQTTLens - Allows us to see the messages getting passed
- Postman - Allows us to send http requests to the RESTful interface
- Chrome - allows us to run an in browser visual demonstration

SIMULATION OF PLATFORM – COMPONENTS

- Client SDK - For using with various platforms (index.js)
  - Browsers
  - Node.js
- Browser - For demonstration
  - demo.js - implementation of the Client SDK

CLIENT SDK (INDEX.JS)

```
mqtt = require('mqtt')
```

MQTT Library

```
/**
 * [Client description]
 * @param {[type]} ID [description]
 */
function Client(ID, JSON){
      //check with the server if this is a valid id
      //also needs some kind of authentication
      if(ID){
            this.id = ID
      }
      else{
            console.log("You have initialized the client without the ID")
      }
}
```

CLIENT SDK (INDEX.JS)

```
Client.prototype = {
      //this is the location of the MQTT server right now
      mqtt_options: {
        clientId: 'mqttjs_' + Math.random().toString(16).substr(2, 8),
        protocolId: 'MQTT',
      },
      mqtt_uri: 'ws://localhost:1884',
      /**
       * Initializes the MQTT connection up with the presence and id topic subscribed to
       * @return none
       */
      init: function(){
            if(this.id){
                  this.client = mqtt.connect(this.mqtt_uri, this.mqtt_options)
                  var C = this.client
                  console.log(this.client)

                  //we have to assign the Client the function because of the
                  //scoped nature of Node that way we will have access to the function in the callbacks

                  C.functions = this.functions

                  C.on('connect', function () {
                    C.subscribe('presence')
                    C.publish('presence', 'Hello mqtt')
                  })

                  C.on('message', this.onMessage)
                  //NOT SURE IF THIS WILL BE A SECURITY NIGHTMARE
                  this.subscribe(this.id + "/#") //subscribe to all id channels
            }
            else{
                  console.log('Please add an ID')
            }
},
```

MQTT WebSocket URL on port 1884

CLIENT SDK (INDEX.JS)

```
Client.prototype = {
      //this is the location of the MQTT server right now
      mqtt_options: {
        clientId: 'mqttjs_' + Math.random().toString(16).substr(2, 8),
        protocolId: 'MQTT',
      },
      mqtt_uri: 'ws://localhost:1884',
      /**
       * Initializes the MQTT connection up with the presence and id topic subscribed to
       * @return none
       */
      init: function(){
            if(this.id){
                  this.client = mqtt.connect(this.mqtt_uri, this.mqtt_options)
                  var C = this.client

                  console.log(this.client)

                  //we have to assign the Client the function because of the
                  //scoped nature of Node that way we will have access to the function in the callbacks

                  C.functions = this.functions

                  C.on('connect', function () {
                    C.subscribe('presence')
                    C.publish('presence', 'Hello mqtt')
                  })

                  C.on('message', this.onMessage)
                  //NOT SURE IF THIS WILL BE A SECURITY NIGHTMARE
                  this.subscribe(this.id + "/#") //subscribe to all id channels
            }
            else{
                  console.log('Please add an ID')
            }
},
```

Connect

CLIENT SDK (INDEX.JS)

```
Client.prototype = {
      //this is the location of the MQTT server right now
      mqtt_options: {
        clientId: 'mqttjs_' + Math.random().toString(16).substr(2, 8),
        protocolId: 'MQTT',
      },
      mqtt_uri: 'ws://localhost:1884',
      /**
       * Initializes the MQTT connection up with the presence and id topic subscribed to
       * @return none
       */
      init: function(){
            if(this.id){
                  this.client = mqtt.connect(this.mqtt_uri, this.mqtt_options)
                  var C = this.client
                  console.log(this.client)

                  //we have to assign the Client the function because of the
                  //scoped nature of Node that way we will have access to the function in the callbacks

                  C.functions = this.functions

                  C.on('connect', function () {
                    C.subscribe('presence')
                    C.publish('presence', 'Hello mqtt')
                  })

                  C.on('message', this.onMessage)
                  //NOT SURE IF THIS WILL BE A SECURITY NIGHTMARE
                  this.subscribe(this.id + "/#") //subscribe to all id channels
            }
            else{
                  console.log('Please add an ID')
            }
},
```

Important!

CLIENT SDK (INDEX.JS)

```
      /**
       * subscribes the client to a channel
       * @param  {string} topic the topic to be subscribed to
       */
      subscribe: function(topic){
            this.client.subscribe(topic)
      },
      /**
       * Publishes a message to the MQTT server for propogation
       * @param  {string} topic   the topic to be published to
       * @param  {string} message the message to be published
       */
      publish: function(topic, message){
            this.client.publish(topic,message)
      },
      /**
       * the callback function for MQTT on message
       * @param  {string} topic   the topic of the message
       * @param  {string} message the message byte string
       */
      onMessage: function(topic, message){
            var url_arr = topic.split('/')

            if(url_arr[1] === 'transition'){
                  // console.log(url_arr[2])
                  this.functions[url_arr[2]](message.toString())
                  return
            }
            // we will probably add more functionality here
            // this will be a large function
      }
}

//for browsers?
c = Client
//for node
module.exports = Client
```

Important!

CLIENT SDK (INDEX.JS)

```
        /**
         * subscribes the client to a channel
         * @param  {string} topic the topic to be subscribed to
         */
        subscribe: function(topic){
              this.client.subscribe(topic)
        },
        /**
         * Publishes a message to the MQTT server for propogation
         * @param  {string} topic   the topic to be published to
         * @param  {string} message the message to be published
         */
        publish: function(topic, message){
              this.client.publish(topic,message)
        },
        /**
         * the callback function for MQTT on message
         * @param  {string} topic   the topic of the message
         * @param  {string} message the message byte string
         */
        onMessage: function(topic, message){
              var url_arr = topic.split('/')

              if(url_arr[1] === 'transition'){
                    // console.log(url_arr[2])
                    this.functions[url_arr[2]](message.toString())
                    return
              }
              // we will probably add more functionality here
              // this will be a large function
        }
}

//for browsers?
c = Client
//for node
module.exports = Client
```

Important!

DEMONSTRATION FILE (DEMO.JS)

```
var machine = {

}

var client = new c('aasdkfjhaskjdhf', machine) //fake id

//these functions work because it is scoped
client.functions = {
      sayHi: function(){
            console.log('HELLO WORLD!');
      },
      sayNo: function(){
            console.log('I DONT WANT TO TALK!');
      },
      sayMessage: function(message){
            console.log('message')
      },
      makeBox: function(message){
            $(‘body').append('
                <div style="background-color:pink; display:block; width:50px; height:50px;”>
                '+message+'</div>')
      }
}

client.init()    //start the server
```

Instantiate the client

DEMONSTRATION FILE (DEMO.JS)

```
var machine = {

}

var client = new c('aasdkfjhaskjdhf', machine) //fake id

//these functions work because it is scoped
client.functions = {
      sayHi: function(){
            console.log('HELLO WORLD!');
      },
      sayNo: function(){
            console.log('I DONT WANT TO TALK!');
      },
      sayMessage: function(message){
            console.log('message')
      },
      makeBox: function(message){
            $(‘body').append('
                <div style="background-color:pink; display:block; width:50px; height:50px;”>
                '+message+'</div>')
      }
}

client.init()    //start the server
```

Instantiate the client

Define the functions

DEMONSTRATION FILE (DEMO.JS)

```
var machine = {

}

var client = new c('aasdkfjhaskjdhf', machine) //fake id

//these functions work because it is scoped
client.functions = {
      sayHi: function(){
            console.log('HELLO WORLD!');
      },
      sayNo: function(){
            console.log('I DONT WANT TO TALK!');
      },
      sayMessage: function(message){
            console.log('message')
      },
      makeBox: function(message){
            $(‘body').append('
                <div style="background-color:pink; display:block; width:50px; height:50px;”>
                '+message+'</div>')
      }
}

client.init()    //start the server
```

Instantiate the client

Define the functions

Connect

PROPOSED CAPSTONE DEMONSTRATION

- Embedding framework logic in various applications
  - Remote control simple music player application
    - Cross platform compatibility
  - Controlling IoT device
    - Demonstrate dependency feature