PointMap

A Geospatial Mapping Tool

# Logging - Developer Doc

Christopher Meyer

April 24, 2019

## Revision Table

| Ver | Date | Amendement | Author |
|---|---|---|---|
| v1.0 | 2019/04/24 | Initialization | Chris Meyer |

# Table Of Contents

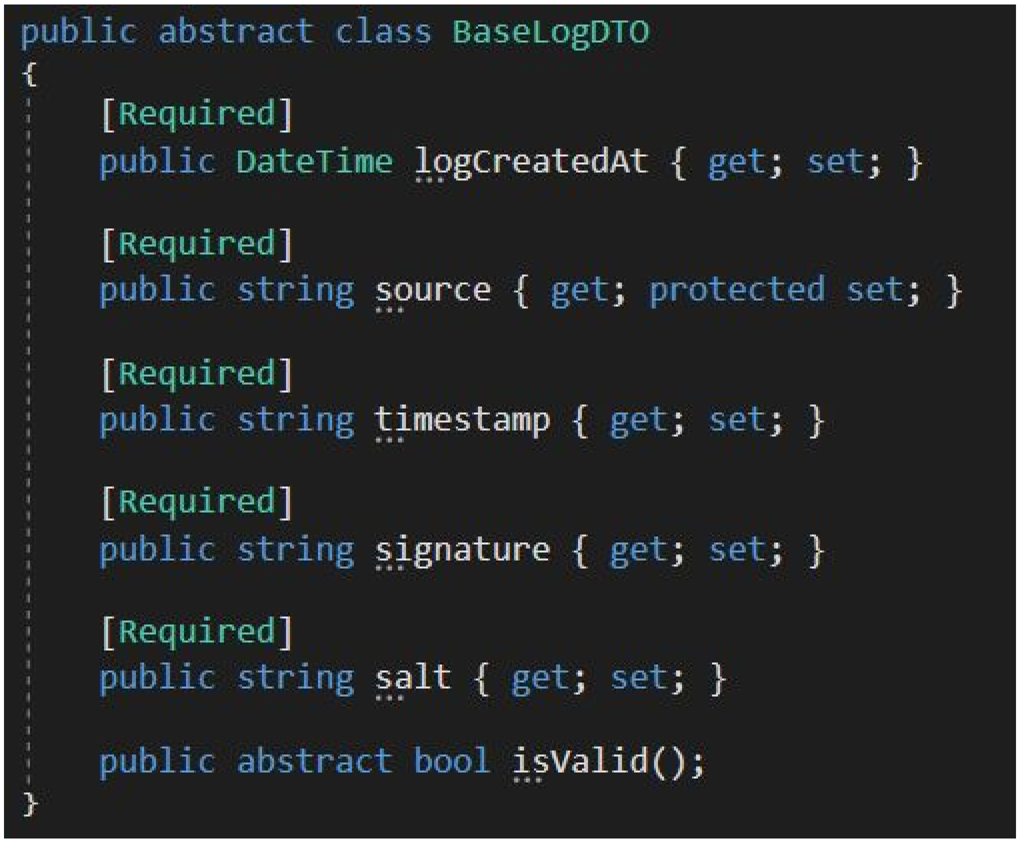
# Implementation

Logging is separated between two major parts, the Logging Server and the Logging Service, Project and DTO on the .Net Backend.

# ASP.Net Backend

## BaseLogDTO

The BaseLogDTO(BLDTO) is used to transfer, validate, and store information used for the creation of a log. The base log DTO holds information and methods that all derived log types must contain.

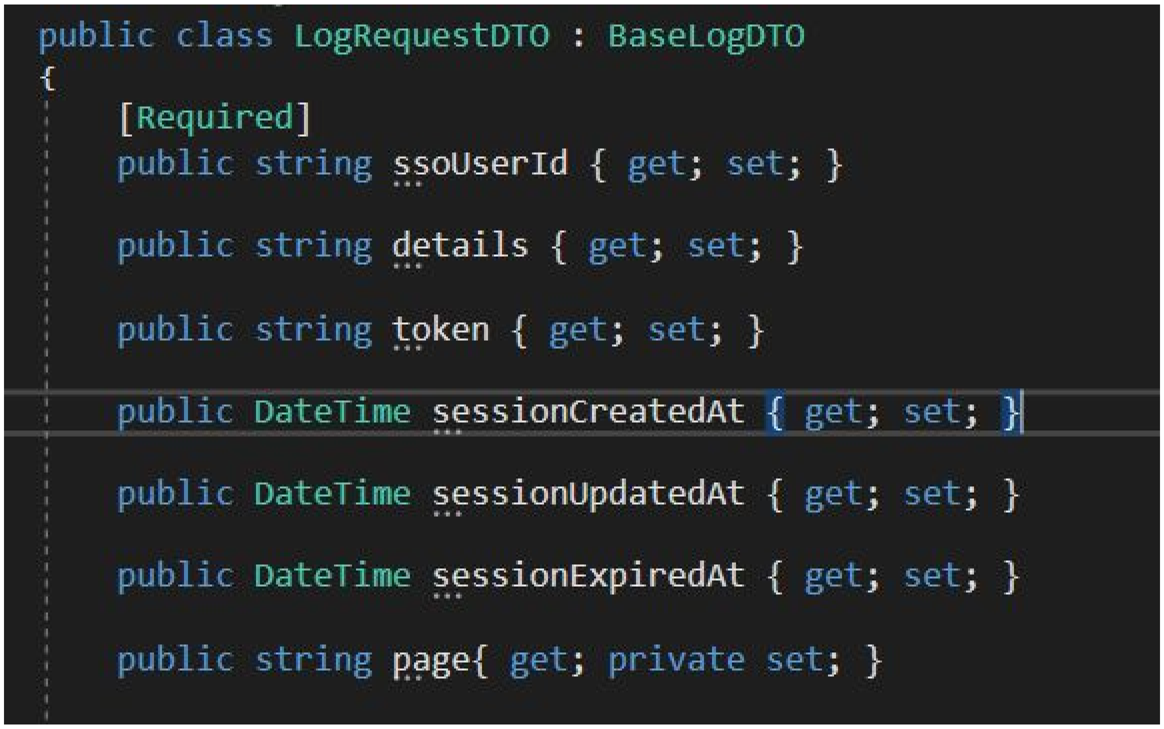


```
public abstract class BaseLogDTO
{
    [Required]
    public DateTime logCreatedAt { get; set; }

    [Required]
    public string source { get; protected set; }

    [Required]
    public string timestamp { get; set; }

    [Required]
    public string signature { get; set; }

    [Required]
    public string salt { get; set; }

    public abstract bool isValid();
}
```

- The required fields are data that will be sent in the body of the POST request to the Logging Express server that are either needed for the authentication of the POST request.
- The method IsValid is required for the validation of all logs

## LogRequestDTO

LogRequestDTO inherits BaseLogDTO, and is created for analytical data. This object holds required and non-required data fields meant for analyzing information about the PointMap application. Holding non-required fields allows for an easy extension and addition of information to this object.

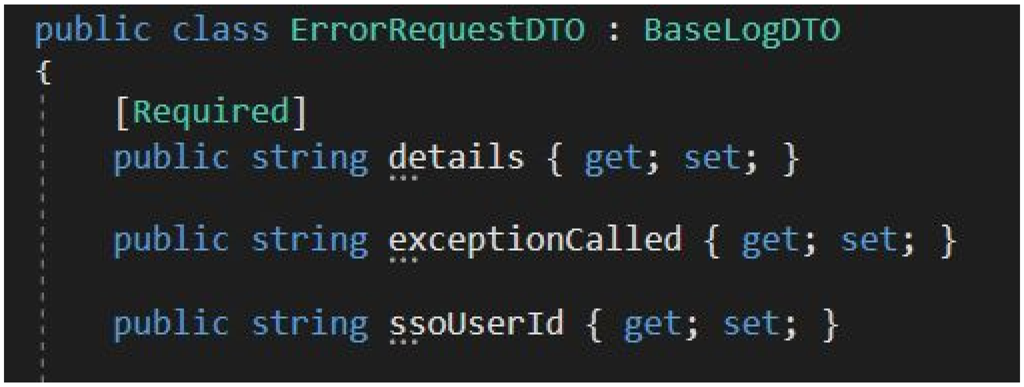


```
public class LogRequestDTO : BaseLogDTO
{
    [Required]
    public string ssoUserId { get; set; }

    public string details { get; set; }

    public string token { get; set; }

    public DateTime sessionCreatedAt { get; set; }

    public DateTime sessionUpdatedAt { get; set; }

    public DateTime sessionExpiredAt { get; set; }

    public string page{ get; private set; }
```

This class also holds a variety of constructors and set methods for the value of source and page. Source and page may only hold a value from an enumeration defined in the constants class for querying purposes.

## ErrorRequestDTO

ErrorRequest inherits BaseLogDTO, and is created for error capturing. This object holds required and non-required data fields meant for error tracking in the PointMap application. Holding non-required fields allows for an easy extension and addition of information to this object.

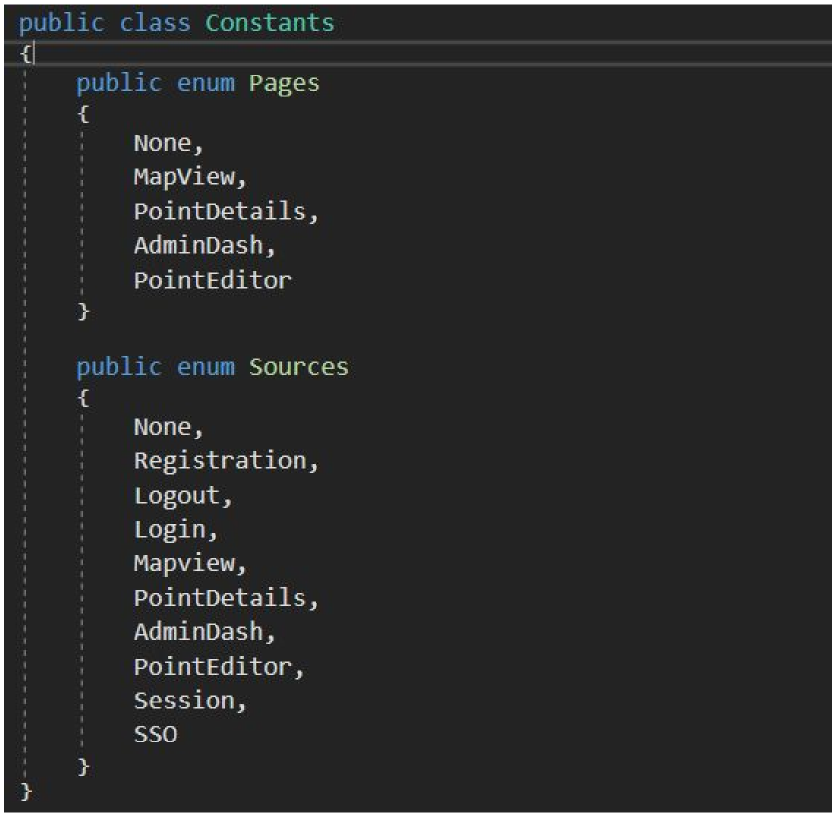


```
public class ErrorRequestDTO : BaseLogDTO
{
    [Required]
    public string details { get; set; }

    public string exceptionCalled { get; set; }

    public string ssoUserId { get; set; }
```

In the same way as LogRequestDTO this class holds a variety of constructors and a set method for the value of source.

## Constants

Constants is a class meant to hold constant information regarding enumerations or other data properties. This class allows for an easy adjustment and extensibility of enumerations and fields by being a singular data holder for the DTOs.



```
public class Constants
{
    public enum Pages
    {
        None,
        MapView,
        PointDetails,
        AdminDash,
        PointEditor
    }

    public enum Sources
    {
        None,
        Registration,
        Logout,
        Login,
        Mapview,
        PointDetails,
        AdminDash,
        PointEditor,
        Session,
        SSO
    }
}
```

## Logger

The logger is located inside a separate Logging project, so that it may be invoked at any time. Logger is in charge of beginning the process of sending BaseLogDTO (BLDTO) derived objects to the Logging Server and validating the contents of the BLDTO derived object by invoking the method isValid() contained inside the BLDTO derived classes.

If the BLDTO object is validated the Logger invokes a method to generate the timestamp, salt and signature for the BLDTO. The signature is generated using a SHA256 hasher from the LoggingService class using the following plaintext contained in the BLDTO and a shared secret.

```
string plaintext = "timestamp=" + timestamp + ";salt=" + salt;
```

The outputted signature hash is produced in BASE64, using the System.Security library.

After the timestamp salt and signature has been appended to the BLDTO, the object is passed to LoggingService to transmit the request.

## LoggingService

The LoggingService is located in the service layer and contains three main functionalities.

1. Performs signature and salt generation for authentication of requests between .Net Backend and Logging Server
2. Forms the content and sends the HTTPS request to the Logging Server
3. Notifies system admin vi email after 100 failed logs

## Logging Server

The Logging server contains two routes, as described in the API documentation in the Design Document.

- POST route for storing a new log
- POST route for storing a new error

The Logging server additionally contains three main directories that are related to logging in addition to the main class, app.js.

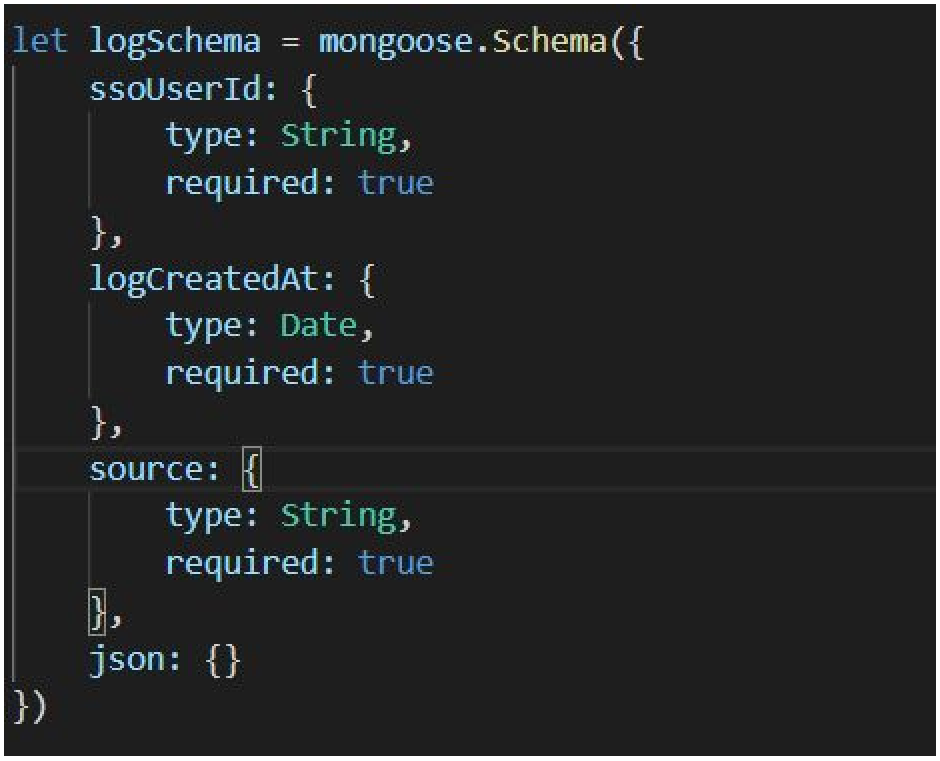
- Services
    - Provides helpful services for the storing and authorization of requests
- Models
    - Contains the error.js and log.js schema, which formats the building of the logs/errors
- Router
    - Contains the POST routes ‘/log’ and ‘/error’ for which logging requests are sent to

## Models

The following section reviews the models directory.

### Log Schema

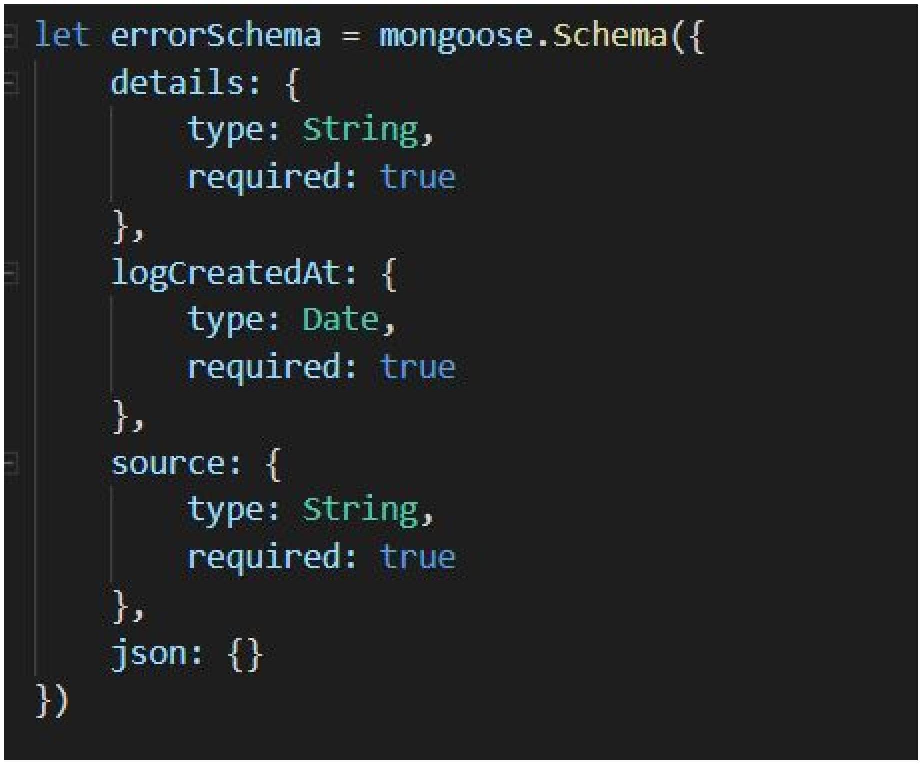
The Log schema is defined as follows;


```
let logSchema = mongoose.Schema({
    ssoUserId: {
        type: String,
        required: true
    },
    logCreatedAt: {
        type: Date,
        required: true
    },
    source: {
        type: String,
        required: true
    },
    json: {}
})
```

All fields except those regarding authentication that are required in the LogRequestDTO are also required for the Log schema, additionally do to the flexibility of the data storage in MongoDB all non-required fields are able to be saved as nested objects in the "json" field. This allows for reusability of the logging schema as it is not limited to a set of defined fields.

## Error Schema

The error schema is defined as follows



```
let errorSchema = mongoose.Schema({
    details: {
        type: String,
        required: true
    },
    logCreatedAt: {
        type: Date,
        required: true
    },
    source: {
        type: String,
        required: true
    },
    json: {}
})
```

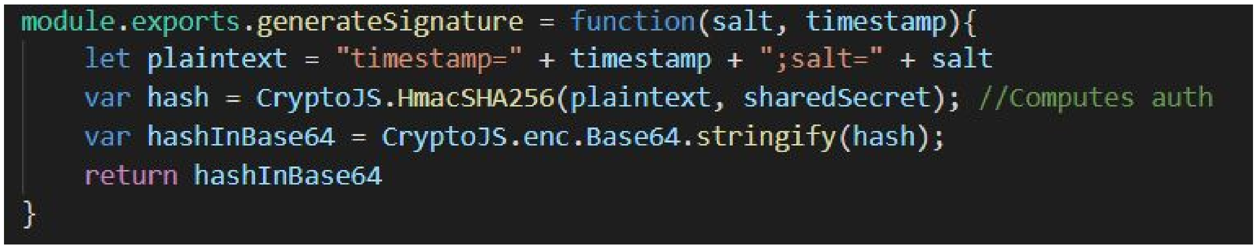
The error schema is created in the same way as the log schema, with different fields that are more pertinent to errors. All non-required fields are stored in the json object.

## Services

The following section reviews the services directory

### AuthorizationService

- Function generateSignature(salt, timestamp)
    - Generates the signature required to verify any incoming requests.

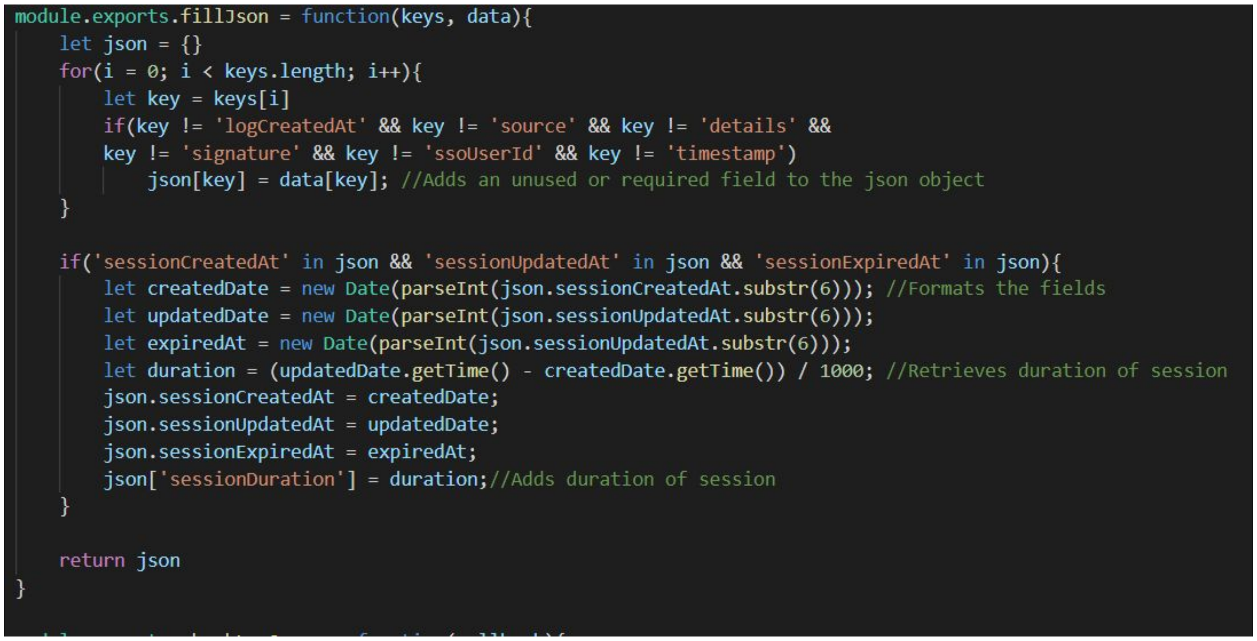


```
module.exports.generateSignature = function(salt, timestamp){
    let plaintext = "timestamp=" + timestamp + ";salt=" + salt
    var hash = CryptoJS.HmacSHA256(plaintext, sharedSecret); //Computes auth
    var hashInBase64 = CryptoJS.enc.Base64.stringify(hash);
    return hashInBase64
}
```

### loggingService

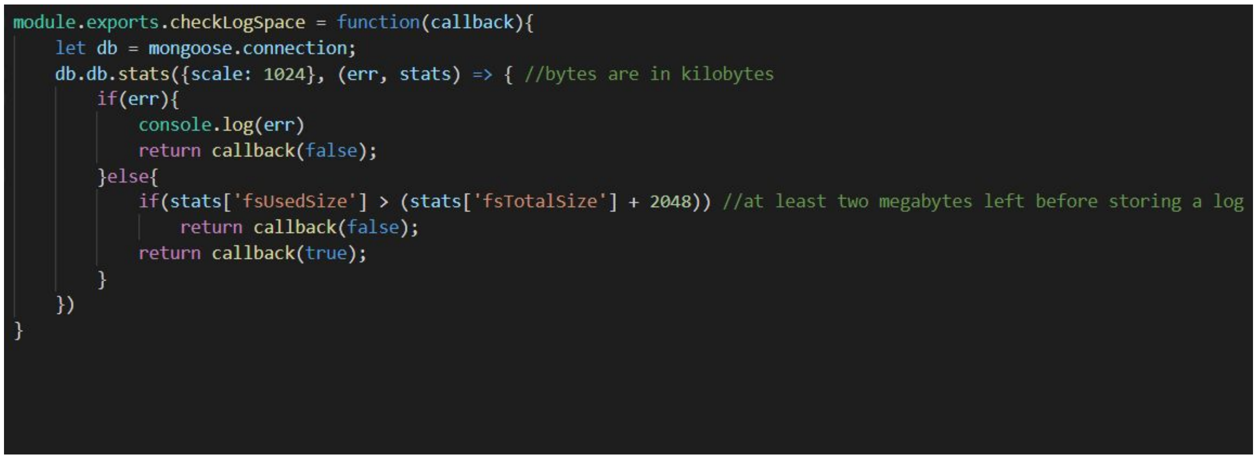
- Function fillJson(keys, data)

        - Returns a Json object with nested information of all non-required fields for a request



```
module.exports.fillJson = function(keys, data){
    let json = {}
    for(i = 0; i < keys.length; i++){
        let key = keys[i]
        if(key != 'logCreatedAt' && key != 'source' && key != 'details' &&
        key != 'signature' && key != 'ssoUserId' && key != 'timestamp')
            json[key] = data[key]; //Adds an unused or required field to the json object
    }

    if('sessionCreatedAt' in json && 'sessionUpdatedAt' in json && 'sessionExpiredAt' in json){
        let createdDate = new Date(parseInt(json.sessionCreatedAt.substr(6))); //Formats the fields
        let updatedDate = new Date(parseInt(json.sessionUpdatedAt.substr(6)));
        let expiredAt = new Date(parseInt(json.sessionUpdatedAt.substr(6)));
        let duration = (updatedDate.getTime() - createdDate.getTime()) / 1000; //Retrieves duration of session
        json.sessionCreatedAt = createdDate;
        json.sessionUpdatedAt = updatedDate;
        json.sessionExpiredAt = expiredAt;
        json['sessionDuration'] = duration;//Adds duration of session
    }

    return json
}
```

- Function checkLogSpace
    - Checks the mongodb database to check if there is adequate storage space to store any incoming data. After storage is less than 2MB errors and logs will not be allowed to be stored.



```
module.exports.checkLogSpace = function(callback){
    let db = mongoose.connection;
    db.db.stats({scale: 1024}, (err, stats) => { //bytes are in kilobytes
        if(err){
            console.log(err)
            return callback(false);
        }else{
            if(stats['fsUsedSize'] > (stats['fsTotalSize'] + 2048)) //at least two megabytes left before storing a log
                return callback(false);
            return callback(true);
        }
    })
}
```

## Router

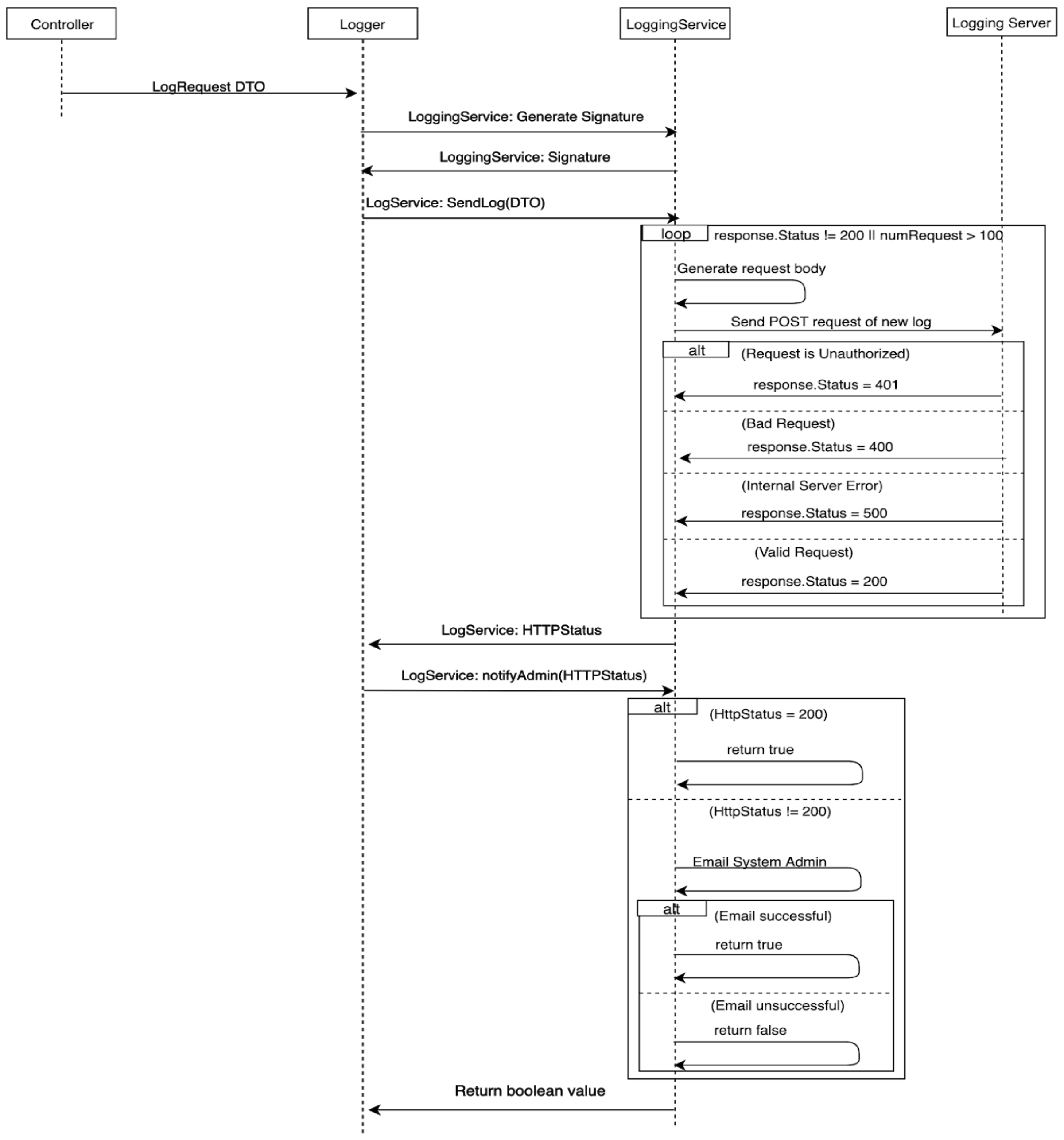
The following section reviews the Router directory

## Routes

- The routes directory holds two API available to be used for the creation of logs and errors
    - POST ‘/log’
    - POST ‘/error’
- These API are covered in detail in the Logging API Doc

## Logging WorkFlow Diagram

The following diagram show the logic flow of communication between the logging service in the PointMap Backend, and the Logging Server.


Controller
Logger
LoggingService
Logging Server
LogRequest DTO
LoggingService: Generate Signature
LoggingService: Signature
LogService: SendLog(DTO)
loop
response.Status != 200 II numRequest > 100
Generate request body
Send POST request of new log
alt
(Request is Unauthorized)
response.Status = 401
(Bad Request)
response.Status = 400
(Internal Server Error)
response.Status = 500
(Valid Request)
response.Status = 200
LogService: HTTPStatus
LogService: notifyAdmin(HTTPStatus)
alt
(HttpStatus = 200)
return true
(HttpStatus != 200)
Email System Admin
alt
(Email successful)
return true
(Email unsuccessful)
return false
Return boolean value