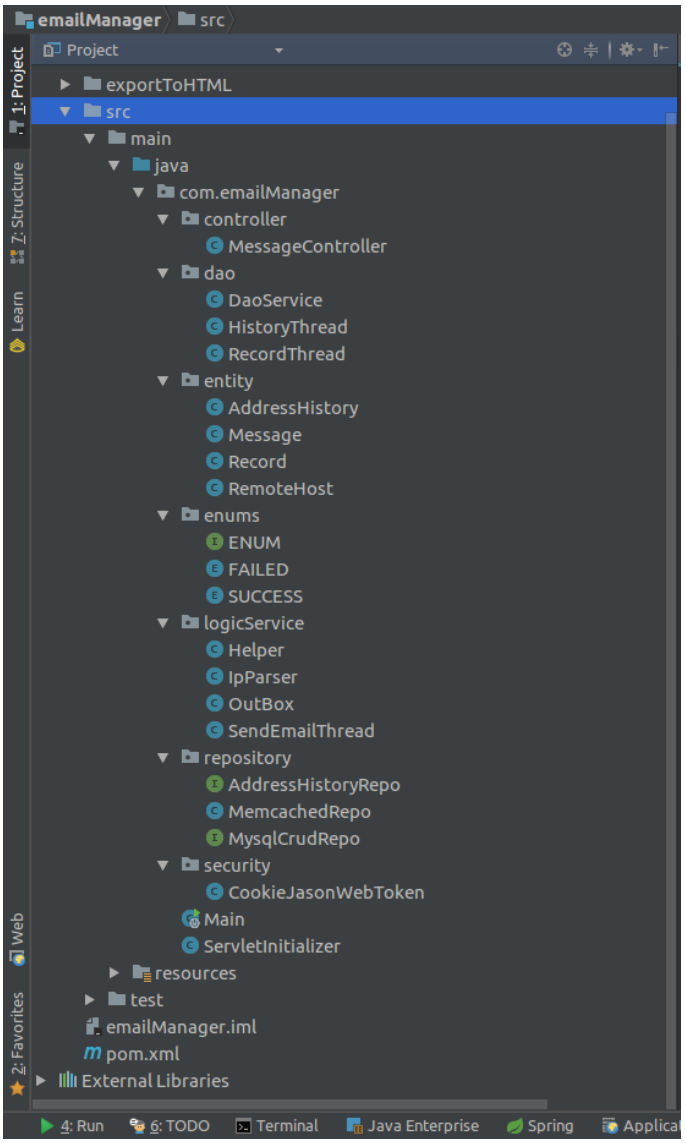
**********Project Structure**********



**********emialManager Layer**********

Main.java

```java
package com.emailManager;

import net.spy.memcached.ConnectionFactory;
import net.spy.memcached.ConnectionFactoryBuilder;
import net.spy.memcached.FailureMode;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import com.btmatthews.springboot.memcached.EnableMemcached;
import org.springframework.context.annotation.Bean;

/**
 * Springboot application main class.
 */
@SpringBootApplication
@EnableMemcached
public class Main{
    public static void main(String[] args){
        SpringApplication.run(Main.class, args);
    }


    @Bean
```

```
23      public ConnectionFactory memcachedConnection() {
24          return new ConnectionFactoryBuilder()
25                  .setDaemon(true)
26                  .setFailureMode(FailureMode.Cancel)
27                  .build();
28      }
29  }
30
```

ServletInitializer.java

```
1   package com.emailManager;
2
3   import org.springframework.boot.builder.SpringApplicationBuilder;
4   import org.springframework.boot.web.support.SpringBootServletInitializer;
5
6   /**
7    * Springboot servlet initializer class.
8    */
9   public class ServletInitializer extends SpringBootServletInitializer {
10      @Override
11      public SpringApplicationBuilder configure(SpringApplicationBuilder springApplicationBuilder){
12          return springApplicationBuilder.sources(Main.class);
13
14      }
15  }
16
```

**********controller Layer**********

MessageController.java

```
1   package com.emailManager.controller;
2
3   import com.emailManager.enums.ENUM;
4   import com.emailManager.dao.DaoService;
5   import com.emailManager.logicService.*;
6   import com.emailManager.logicService.SendEmailThread;
7   import com.emailManager.entity.*;
8   import com.emailManager.enums.SUCCESS;
9   import com.google.common.collect.ImmutableMap;
10  import org.springframework.beans.factory.annotation.Autowired;
11  import org.springframework.http.MediaType;
12  import org.springframework.mail.javamail.JavaMailSender;
13  import org.springframework.web.bind.annotation.*;
14
15  import javax.servlet.http.HttpServletRequest;
16  import javax.servlet.http.HttpServletResponse;
17
18  /**
19   * Microservice RESTful API: to receive and reply to http request calls remotely.
20   */
21  @CrossOrigin
22  @RestController
23  public class MessageController {
24      @Autowired
25      public JavaMailSender emailSender;
26      @Autowired
27      DaoService daoService;
28      @Autowired
29      Helper helper;
30      private AddressHistory emailHistory;
31      ENUM enumMsg;
32
33      /**
34       * @param request: HttpServletRequest
35       * @param message: {"email":"*","name":"*","body":"*"} in JSON format
36       * @return {"JSON result message"}
37       */
38      @CrossOrigin
39      @RequestMapping(value = "/email" ,consumes = MediaType.APPLICATION_JSON_VALUE ,method = {RequestMethod.POST,RequestMethod.GET})
40      public Object email(HttpServletRequest request, @RequestBody Message message, HttpServletResponse response) {
41          if((enumMsg = helper.validate(request, message)) instanceof SUCCESS){
42
43              emailHistory = daoService.getEmailHistory(message.getEmail());
44              OutBox outBox = new OutBox(emailSender, message);
45
46              if(emailHistory == null){
47                  enumMsg = outBox.sendEmail();
48              }
49              else if((enumMsg = emailHistory.dailyMaxMsg()) instanceof SUCCESS){
50                      new Thread(new SendEmailThread(outBox)).start();
51              }
52
53              if(enumMsg instanceof SUCCESS){
54                  daoService.updateDB(request, emailHistory, message);
55                  helper.responseAddJwtToken(request, response);
56              }
57          }
58          return helper.response(enumMsg);
59      }
60
61      /**
62       * You could test this micro service from your browser
```

```
63          * @return
64          */
65         @RequestMapping(value = "/", produces = MediaType.APPLICATION_JSON_VALUE )
66         public Object browserTest(){
67             return ImmutableMap.<String, Object>builder().
68                     put(SUCCESS.STATUS.toString(), SUCCESS.MICRO_SERVICE_ON.toString()).build();
69         }
70
71     }
72
73
74
75
```

**********repository Layer**********

MemcachedRepo.java

```
1    package com.emailManager.repository;
2
3    import com.emailManager.entity.AddressHistory;
4    import com.emailManager.enums.FAILED;
5    import net.spy.memcached.MemcachedClient;
6    import org.springframework.stereotype.Service;
7    import java.io.IOException;
8    import java.net.InetSocketAddress;
9
10
11   /**
12    * This class establishes connection to the local Memcached,
13    * in order to store and quick retrieve objects
14    */
15   @Service
16   public class MemcachedRepo {
17       private MemcachedClient memcachedClient;
18
19       public MemcachedRepo(){
20           connect();
21       }
22
23       public void connect(){
24           try {
25               this.memcachedClient  = new MemcachedClient(new InetSocketAddress("127.0.0.1", 11211));
26           } catch (IOException e) {
27               System.out.println(FAILED.MEM_CACHED_ERROR.toString());
28           }
29       }
30
31       public void saveAddressHistory(AddressHistory addressHistory){
32           if(addressHistory != null){
33               memcachedClient.set(addressHistory.getEmail(),10000, addressHistory);
34           }
35       }
36
37       public AddressHistory getAddressHistory(String emailAddress){
38           return (AddressHistory)memcachedClient.get(emailAddress);
39       }
40
41       public void deleteAddressHistory(String emailAddress) {
42           memcachedClient.delete(emailAddress);
43
44       }
45
46       public MemcachedClient getMemcachedClient() {
47           return memcachedClient;
48       }
49
50       public void setMemcachedClient(MemcachedClient memcachedClient) {
51           this.memcachedClient = memcachedClient;
52       }
53   }
54
```

AddressHistoryRepo.java

```
1    package com.emailManager.repository;
2
3    import com.emailManager.entity.AddressHistory;
4    import org.springframework.data.repository.CrudRepository;
5    import org.springframework.stereotype.Service;
6
7    @Service
8    public interface AddressHistoryRepo extends CrudRepository<AddressHistory, String> {
9        /**
10        * This area is for JPA custom methods
11        */
12   }
13
```

MysqlCrudRepo.java

```
1    package com.emailManager.repository;
2    import com.emailManager.entity.Message;
3    import com.emailManager.entity.Record;
```

```
4   import org.springframework.data.repository.CrudRepository;
5   import org.springframework.stereotype.Service;
6
7
8   @Service
9   public interface MysqlCrudRepo extends CrudRepository<Record, Message> {
10      /**
11       * JPA custom methods
12       */
13
14  }
15
```

**********security Layer**********

CookieJasonWebToken.java

```
1   package com.emailManager.security;
2
3
4   import com.emailManager.enums.SUCCESS;
5   import com.emailManager.entity.RemoteHost;
6   import com.fasterxml.jackson.databind.ObjectMapper;
7   import io.jsonwebtoken.Jwts;
8   import io.jsonwebtoken.SignatureAlgorithm;
9   import org.springframework.stereotype.Service;
10
11  import javax.servlet.http.Cookie;
12  import javax.servlet.http.HttpServletRequest;
13  import java.time.Instant;
14  import java.util.Date;
15
16  @Service
17  public class CookieJasonWebToken {
18      private static final long serialversionUID = 121345637L;
19      private final String JWT_HEADER = SUCCESS.TOKEN_HEADER.toString();
20      private final String KEY = SUCCESS.TOKEN_KEY.toString();
21
22      /**
23       * @param remoteHost NOT NULL
24       * @return a cookie with a Jwt token embedded
25       */
26      public Cookie creatCookieJwt(RemoteHost remoteHost){
27          String jwt = creatJwtToken(remoteHost);
28          return new Cookie(JWT_HEADER, jwt);
29
30      }
31
32      /**
33       * @param remoteHost NOT NULL
34       * @return a JwtToken string with a remoteHost obj embedded.
35       */
36      public String creatJwtToken(RemoteHost remoteHost){
37          String Jwt = Jwts.builder().claim(JWT_HEADER, remoteHost).signWith(SignatureAlgorithm.HS256, KEY)
38                  .setIssuedAt(new Date(Instant.now().toEpochMilli())).compact();
39          return Jwt;
40      }
41
42      /**
43       * @param jwts
44       * @return the RemoteHost obj contained in the jwt string
45       */
46      public RemoteHost getJwtRemoteHost(String jwts){
47          try{
48              Object token = Jwts.parser().setSigningKey(KEY).parseClaimsJws(jwts).getBody().get(JWT_HEADER);
49
50             return (new ObjectMapper()).convertValue(token, RemoteHost.class);
51          }catch (Exception e){}
52          return null;
53      }
54
55      /**
56       * @param request
57       * @return the RemoteHost obj contained in the Cookie (Http request param)
58       */
59      public RemoteHost getCookieRemoteHost(HttpServletRequest request){
60          Cookie cookies[] = request.getCookies();
61          String jwts, cookieName;
62          RemoteHost remoteHost;
63          if(cookies != null){
64              for(Cookie c: cookies){
65                  cookieName = c.getName();
66                  jwts = c.getValue();
67                  if(cookieName !=null && cookieName.equals(JWT_HEADER)) {
68                      remoteHost = getJwtRemoteHost(jwts);
69                      if(remoteHost != null)
70                          return remoteHost;
71                  }
72              }
73          }
74          return null;
75      }
76
77      /**
78       * @return the JWT header string
```

```
79       */
80      public String getJWT_HEADER() {
81          return JWT_HEADER;
82      }
83  }
84
```

**********dao Layer**********

DaoService.java

```
1   package com.emailManager.dao;
2
3   import com.emailManager.entity.AddressHistory;
4   import com.emailManager.entity.Message;
5   import com.emailManager.logicService.IpParser;
6   import com.emailManager.repository.AddressHistoryRepo;
7   import com.emailManager.repository.MemcachedRepo;
8   import com.emailManager.repository.MysqlCrudRepo;
9   import com.emailManager.security.CookieJasonWebToken;
10  import org.springframework.beans.factory.annotation.Autowired;
11  import org.springframework.stereotype.Service;
12  import javax.servlet.http.HttpServletRequest;
13
14  @Service
15  public class DaoService {
16
17      @Autowired
18      private AddressHistoryRepo addressHistoryRepo;
19      @Autowired
20      private MemcachedRepo memcachedRepo;
21      @Autowired
22      private MysqlCrudRepo mysqlCrudRepo;
23      @Autowired
24      IpParser ipParser;
25      @Autowired
26      CookieJasonWebToken cookieJasonWebToken;
27
28      private String remoteAddr;
29      private String remoteHost;
30
31      /**
32       * if the emailHistory Key exists in mySql db then the threads run Asynchronous/parallel,
33       * else t2 must wait for t1 to create the corresponding key due to a REFERENCE constraint.
34       * @param request
35       * @param emailHistory
36       * @param message
37       */
38      public void updateDB(HttpServletRequest request, AddressHistory emailHistory, Message message) {
39          Thread t1 = saveEmailHistory(message, emailHistory);
40          Thread t2 = saveMessage(request, message);
41          t1.start();
42          if(emailHistory == null){
43              try {
44                  t1.join();
45              } catch (InterruptedException e) {
46                  e.printStackTrace();
47              }
48          }
49          t2.start();
50      }
51
52      /**
53       * @param message
54       * @param emailHistory
55       * @return Thread object to update MySql db
56       */
57      public Thread saveEmailHistory( Message message, AddressHistory emailHistory) {
58          AddressHistory eH = (emailHistory!=null)?emailHistory:new AddressHistory(message.getEmail(), message.getName());
59          return new Thread(new HistoryThread(memcachedRepo, addressHistoryRepo, eH, message));
60      }
61
62      /**
63       * @param request
64       * @param message
65       * @return a Thread obj to update MySql db
66       */
67      public Thread saveMessage(HttpServletRequest request, Message message){
68          this.remoteAddr = ipParser.getClientIpAddress(request);
69          this.remoteHost = request.getRemoteHost();
70          return new Thread(new RecordThread(mysqlCrudRepo, message, remoteAddr, remoteHost));
71      }
72
73
74      /**
75       * Searching for the email history in our local MEMCACHED system,
76       * otherwise it queries MySql db.
77       * @param email
78       */
79      public AddressHistory getEmailHistory(String email) {
80          try{
81              AddressHistory emailHistory = memcachedRepo.getAddressHistory(email);
82              if(emailHistory == null){
83                  emailHistory = addressHistoryRepo.findOne(email);
84              }
```

```
85              return emailHistory;
86          }catch (Exception e){}
87          return null;
88      }
89
90
91  }
92
```

HistoryThread.java

```
1   package com.emailManager.dao;
2   import com.emailManager.entity.AddressHistory;
3   import com.emailManager.entity.Message;
4   import com.emailManager.repository.AddressHistoryRepo;
5   import com.emailManager.repository.MemcachedRepo;
6
7   public class HistoryThread implements Runnable{
8
9       private MemcachedRepo memcachedRepo;
10      private AddressHistoryRepo addrHistRepo;
11      private AddressHistory addrHistory;
12      public String email;
13      private String name;
14
15      /**
16       * @param memcached
17       * @param addrRepo
18       * @param addrHist
19       * @param msg
20       */
21      public HistoryThread(MemcachedRepo memcached, AddressHistoryRepo addrRepo, AddressHistory addrHist, Message msg) {
22          this.memcachedRepo = memcached;
23          this.addrHistRepo = addrRepo;
24          this.addrHistory = addrHist;
25          this.email = msg.getEmail();
26          this.name = msg.getName();
27      }
28
29      /**
30       * addressHistory obj: it is saved locally in Memcached and in the * sql table.
31       */
32      @Override
33      public void run(){
34          try{
35              addrHistory.updateLastVisited();
36              addrHistory.incrementCount();
37
38              memcachedRepo.saveAddressHistory(addrHistory);
39              addrHistRepo.save(addrHistory);
40          }
41          catch (Exception e){
42              e.printStackTrace();
43          }
44      }
45
46  }
47
48
```

RecordThread.java

```
1   package com.emailManager.dao;
2
3   import com.emailManager.entity.Message;
4   import com.emailManager.entity.Record;
5   import com.emailManager.repository.MysqlCrudRepo;
6
7   public class RecordThread implements Runnable {
8       private MysqlCrudRepo mysqlCrudRepo;
9       private Record newRecord;
10      private String remoteAddr;
11      private String remoteHost;
12
13      public RecordThread(MysqlCrudRepo mysqlCrudRepo, Message message, String remoteAddr, String remoteHost) {
14          this.mysqlCrudRepo = mysqlCrudRepo;
15          this.newRecord = new Record();
16          this.newRecord.setMessage(message);
17          this.remoteAddr = remoteAddr;
18          this.remoteHost = remoteHost;
19      }
20
21      /**
22       * Saves an incoming message in MySql db to keep a record.
23       */
24      @Override
25      public void run() {
26          try{
27              newRecord.setRemoteAddr(this.remoteAddr);
28              newRecord.setRemoteHost(this.remoteHost);
29              mysqlCrudRepo.save(newRecord);
30          }
31          catch (Exception e){
32              e.printStackTrace();
33          }
34
35      }
```

```
}
```

**********businessLogic Layer**********

Helper.java

```
package com.emailManager.logicService;

import com.emailManager.enums.ENUM;
import com.emailManager.enums.FAILED;
import com.emailManager.entity.Message;
import com.emailManager.entity.RemoteHost;
import com.emailManager.enums.SUCCESS;
import com.emailManager.security.CookieJasonWebToken;
import com.google.common.collect.ImmutableMap;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Service;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@Service
public class Helper {
    @Value("${properties.HOST_MAX_MSG}")
    private int HOST_MAX_MSG;
    @Value("${properties.todaysMessages}")
    int todaysMessages;
    @Value("${properties.MILLIS}")
    private int MILLIS;
    @Value("${properties.INBOX_DAILY_MAX_MSG}")
    private int INBOX_DAILY_MAX_MSG;
    @Value("${properties.ZERO}")
    private int ZERO;
    @Autowired
    CookieJasonWebToken cookieJasonWebToken;

    /**
     * Helper method to validate the controller's request and message params
     * @param request
     * @param message
     * @return
     */
    public ENUM validate(HttpServletRequest request, Message message){
        if (message == null || message.invalidParams()) {
            return FAILED.INVALID_PARAMETERS;
        }
        else if(hostDailyMaxExceeded(request)){
            return FAILED.REMOTE_HOST_DAILY_MAX_EXCEEDED;
        }

        return SUCCESS.THANKS;
    }


    /**
     * @param request
     * @return true if host max msg exceeded, else false
     */
    public boolean hostDailyMaxExceeded(HttpServletRequest request) {
        String jwts = request.getHeader(cookieJasonWebToken.getJWT_HEADER());
        RemoteHost remoteHost;
        if((remoteHost = cookieJasonWebToken.getJwtRemoteHost(jwts)) == null){
            remoteHost = cookieJasonWebToken.getCookieRemoteHost(request);
        }
        if(remoteHost != null && this.HOST_MAX_MSG <= remoteHost.getTodaysVisits()){
            return true;
        }
       return false;
    }

    /**
     * response: It is a json string wrapper
     * @param msg of type ENUM interface
     * @return {"status":"message"} in JSON format
     */
    public ImmutableMap<String, Object> response(ENUM msg){
        boolean success  = (msg instanceof SUCCESS)?true: false;
        return ImmutableMap.<String, Object>builder().put(SUCCESS.OK.toString(), success).put(SUCCESS.STATUS.toString(), msg.toString()).build();
    }


    /**
     * Adds JwtCookie to the Http response
     * @param request
     * @param response
     */
    public void responseAddJwtToken(HttpServletRequest request, HttpServletResponse response) {
        String header = cookieJasonWebToken.getJWT_HEADER();
        String jwts = request.getHeader(header);
        RemoteHost remoteHost;
        if((remoteHost = cookieJasonWebToken.getJwtRemoteHost(jwts)) == null){
            if((remoteHost = cookieJasonWebToken.getCookieRemoteHost(request))==null)
                remoteHost = new RemoteHost();
        }
```

```
89
90          remoteHost.resetLastVisited();
91          remoteHost.incrementVisits();
92          response.addHeader(header, cookieJasonWebToken.creatJwtToken(remoteHost));
93          response.addCookie(cookieJasonWebToken.creatCookieJwt(remoteHost));
94
95          incrementTodayMax();
96      }
97
98        /**
99       * Increments the todaysVisit variable.
100      */
101     public void incrementTodayMax() {
102         this.todaysMessages++;
103     }
104
105 }
106
```

IpParser.java

```
1   package com.emailManager.logicService;
2
3   import org.springframework.stereotype.Service;
4   import javax.servlet.http.HttpServletRequest;
5
6   /**
7    * it parses the remote user ip address and host name from the HttpServletRequest param
8    */
9   @Service
10  public class IpParser {
11
12      /**
13       * Default constructor
14       */
15      public IpParser(){}
16
17      /**
18       * @param request
19       * @return the remote user Ip address
20       */
21      public String getClientIpAddress(HttpServletRequest request) {
22          String UNKNOWN = "unknown";
23          String ip;
24          for (String header : HEADERS) {
25              ip = request.getHeader(header);
26              if (ip != null && ip.length() != 0 && !UNKNOWN.equalsIgnoreCase(ip)) {
27                  if(ip.contains(",")){
28                      ip = ip.split(",")[0];
29                  }
30                  return ip;
31              }
32          }
33          return request.getRemoteAddr();
34      }
35
36      /**
37       * Possible request.getHeathers(**) params to extract the remoteUser ip address
38       */
39      private static final String[] HEADERS = {
40              "Proxy-Client-IP",
41              "HTTP_X_FORWARDED_FOR",
42              "HTTP_X_FORWARDED",
43              "HTTP_FORWARDED",
44              "REMOTE_ADDR",
45              "HTTP_CLIENT_IP",
46              "X-Forwarded-For"
47      };
48  }
49
```

OutBox.java

```
1   package com.emailManager.logicService;
2
3   import com.emailManager.enums.ENUM;
4   import com.emailManager.enums.FAILED;
5   import com.emailManager.entity.Message;
6   import com.emailManager.enums.SUCCESS;
7   import org.springframework.mail.SimpleMailMessage;
8   import org.springframework.mail.javamail.JavaMailSender;
9   import org.springframework.mail.javamail.MimeMessageHelper;
10
11  import javax.mail.internet.MimeMessage;
12
13  /**
14   * It encapsulates a message object to be sent out through the JavaMailSender.
15   */
16  public class OutBox {
17      public JavaMailSender emailSender;
18      private Message message;
19      String subject1 = SUCCESS.THANKS.toString() ;
20      String subject2 = SUCCESS.MESSAGE_FROM.toString();
21      String content = SUCCESS.LINK.toString();
22
23      /**
24       * Custom constructor
```

```
25          * @param emailSender
26          * @param message
27          */
28         public OutBox(JavaMailSender emailSender, Message message){
29             this.message = message;
30             this.emailSender = emailSender;
31         }
32 
33         /**
34          * Sends out a confirmation email to the client(sender) user, and one to the owner of the portfolio.
35          * @return
36          */
37         public ENUM sendEmail() {
38             subject2+=message.getEmail();
39             content+="<br><br>"+SUCCESS.MESSAGE_BODY.toString();
40             SimpleMailMessage simpleMailMessage = new SimpleMailMessage();
41             MimeMessage msg = emailSender.createMimeMessage();
42             MimeMessageHelper htmlMessage;
43 
44             try{
45                 msg.setContent(content, "text/html");
46                 htmlMessage = new MimeMessageHelper(msg, false, "utf-8");
47                 htmlMessage.setTo(message.getEmail());
48                 htmlMessage.setSubject(subject1);
49                 emailSender.send(msg);
50 
51                 simpleMailMessage.setTo(SUCCESS.MyEmail.toString());
52                 simpleMailMessage.setSubject(subject2);
53                 simpleMailMessage.setText(message.getBody());
54                 emailSender.send(simpleMailMessage);
55 
56                 return SUCCESS.THANKS;
57             }catch (Exception e){
58                 e.printStackTrace();
59                 return FAILED.INVALID_EMAIL_ADDRESS;
60             }
61         }
62 
63     }
64 
```

SendEmailThread.java

```
1   package com.emailManager.logicService;
2 
3   /**
4    * Multi-threaded object class to send out emails in parallel(Asynchronously)
5    */
6   public class SendEmailThread implements Runnable{
7       private OutBox outBox;
8 
9       /**
10       * Custom constructor
11       * @param outBox
12       */
13      public SendEmailThread(OutBox outBox) {
14          this.outBox = outBox;
15      }
16 
17      /**
18       * Thread run(): Sends out email from the OutBox.send() method
19       */
20      @Override
21      public void run() {
22          outBox.sendEmail();
23      }
24  }
25 
```

**********entity Layer**********

AddressHistory.java

```
1   package com.emailManager.entity;
2 
3   import com.emailManager.enums.ENUM;
4   import com.emailManager.enums.FAILED;
5   import com.emailManager.enums.SUCCESS;
6   import javax.persistence.Entity;
7   import javax.persistence.Id;
8   import javax.persistence.Table;
9   import javax.persistence.Transient;
10  import java.io.Serializable;
11  import java.text.SimpleDateFormat;
12  import java.time.Instant;
13  import java.util.Date;
14 
15  /**
16   * Entity class to be stored as a history record for each email address.
17   */
18  @Entity
19  @Table(name = "addressHistory")
20  public class AddressHistory implements Serializable {
21      @Transient
```

```
22      private static final long serialversionUID = 12374567L;
23      @Id
24      private String email;
25      private String name;
26      private long todaysCount = 0;
27      private long totalCount = 0;
28      private long lastVisited;
29      @Transient
30      private final int SENDER_DAILY_MAX_MSG = 10;
31
32      public AddressHistory() {}
33
34      public AddressHistory(String email, String name) {
35          this.email = email;
36          this.name = name;
37          this.lastVisited = Instant.now().getEpochSecond();
38      }
39
40      public String getName() {
41          return name;
42      }
43
44      public void setName(String name) {
45          this.name = name;
46      }
47
48      public String getEmail() {
49          return email;
50      }
51
52      public void setEmail(String email) {
53          this.email = email;
54      }
55
56      public final int getSENDER_DAILY_MAX_MSG(){
57          return this.SENDER_DAILY_MAX_MSG;
58      }
59
60      public void setTodaysCount(long todaysCount) {
61          this.todaysCount = todaysCount;
62      }
63
64      public long getTotalCount() {
65          return totalCount;
66      }
67
68      public void setTotalCount(long totalCount) {
69          this.totalCount = totalCount;
70      }
71
72      public long getLastVisited() {
73          return lastVisited;
74      }
75
76      public void setLastVisited(long lastVisited) {
77          this.lastVisited = lastVisited;
78      }
79      public void updateLastVisited(){
80          setLastVisited(Instant.now().getEpochSecond());
81      }
82
83      public void incrementCount(){
84          this.todaysCount++;
85          this.totalCount++;
86      }
87
88      public ENUM dailyMaxMsg() {
89          if(this.getTodaysCount() >= this.SENDER_DAILY_MAX_MSG)
90              return FAILED.EMAIL_ADDRESS_DAILY_MAX_EXCEEDED;
91          return SUCCESS.THANKS;
92      }
93
94      public long getTodaysCount() {
95          String pattern = SUCCESS.DATE_FORMAT.toString();
96          SimpleDateFormat simpleDateFormat = new SimpleDateFormat(pattern);
97
98          if(!simpleDateFormat.format(new Date(this.lastVisited * 1000)).equals(
99                  simpleDateFormat.format(new Date(Instant.now().getEpochSecond() * 1000))))
100             {
101                 this.todaysCount = 0;
102             }
103         return todaysCount;
104     }
105 }
106
```

Message.java

```
1   package com.emailManager.entity;
2
3   import com.emailManager.enums.SUCCESS;
4   import com.fasterxml.uuid.Generators;
5   import javax.persistence.Embeddable;
6   import javax.persistence.Transient;
7   import java.io.Serializable;
8   /**
9    * This in an embeddable object, so we can use all its variables as composite
```

```
 * primary keys in our Sql message table.
 */
@Embeddable
public class Message implements Serializable {
    @Transient
    private static final long serialversionUID = 12134567L;
    public String email;
    private String name;
    private String body;
    private String frontEndBackDoorKey;
    private String timeUUID;

    public Message(){
        timeUUID = Generators.timeBasedGenerator().generate().toString();
        }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getBody() {
        return body;
    }

    public void setBody(String body) {
        this.body = body;
    }

    public String getTimeUUID() {
        return timeUUID;
    }

    public void setTimeUUID(String timeUUID) {
        this.timeUUID = timeUUID;
    }

    public String getFrontEndBackDoorKey() {
        return frontEndBackDoorKey;
    }

    public void setFrontEndBackDoorKey(String frontEndBackDoorKey) {
        this.frontEndBackDoorKey = frontEndBackDoorKey;
    }

    public boolean invalidParams(){
        return this.name == null || this.email == null || this.body == null;
    }

    public boolean fromInvalidSource() {
        if(frontEndBackDoorKey == null || !frontEndBackDoorKey.equals(SUCCESS.SECRET_KEY.toString()))
            return true;
        return false;
    }

    @Override
    public String toString() {
        return "Message{" +
                "email='" + email + '\'' +
                ", name='" + name + '\'' +
                ", body='" + body + '\'' +
                ", frontEndBackDoorKey='" + frontEndBackDoorKey + '\'' +
                ", timeStamp=" + timeUUID +
                '}';
    }
}

```

Record.java

```
package com.emailManager.entity;

import javax.persistence.*;
import java.io.Serializable;

/**
 * This is a Record object to be stored in our sql message table,
 * which uses composite primary keys through the embeddable Message object.
 */
@Entity
@Table(name="message")
public class Record implements Serializable {
    @Transient
    private static final long serialversionUID = 1234567L;
    @EmbeddedId
    public Message message;
```

```
17      private String remoteAddr;
18      private String remoteHost;
19
20      public String getRemoteAddr() {
21          return remoteAddr;
22      }
23
24      public void setRemoteAddr(String remoteAddr) {
25          this.remoteAddr = remoteAddr;
26      }
27
28      public boolean invalidParams(){
29          return this.message.invalidParams();
30      }
31
32      public String getRemoteHost() {
33          return remoteHost;
34      }
35
36      public void setRemoteHost(String remoteHost) {
37          this.remoteHost = remoteHost;
38      }
39
40      public Message getMessage() {
41          return message;
42      }
43
44      public void setMessage(Message message) {
45          this.message = message;
46      }
47
48      @Override
49      public String toString() {
50          return "Record{" +
51                  "message=" + message.toString() +
52                  ", remoteAddr='" + remoteAddr + '\'' +
53                  ", remoteHost='" + remoteHost + '\'' +
54                  '}';
55      }
56  }
57
```

RemoteHost.java

```
1   package com.emailManager.entity;
2
3   import com.emailManager.enums.SUCCESS;
4
5   import javax.persistence.Entity;
6   import javax.persistence.*;
7   import java.io.Serializable;
8   import java.text.SimpleDateFormat;
9   import java.time.Instant;
10  import java.util.Date;
11
12  /**
13   * Each object is stored in a Cookie/Jwt token and sent
14   * back to the client side to be stored and retrieved for each visit.
15   */
16  @Entity
17  public class RemoteHost implements Serializable{
18      private static final long serialversionUID = 17234567L;
19      @Id
20      private String remoteIpAddr;
21      private String newIpAddr;
22      private String remoteHost;
23      private long numVisits ;
24      private int todaysVisits;
25      private long firstVisit;
26      private long lastVisited;
27      private int admin;
28
29      public RemoteHost() {
30          this.numVisits = 0;
31          this.todaysVisits = 0;
32          this.firstVisit = Instant.now().getEpochSecond();
33          this.lastVisited = this.firstVisit;
34          this.admin = 0;
35      }
36
37      public int getTodaysVisits() {
38          if(isNewDay()) {
39              this.todaysVisits = 1;
40          }
41          return todaysVisits;
42      }
43
44      public void setTodaysVisits(int todaysVisits) {
45          this.todaysVisits = todaysVisits;
46      }
47
48      public long getLastVisited() {
49          return lastVisited;
50      }
51
52      public void setLastVisited(long lastVisited) {
53          this.lastVisited = lastVisited;
```

```
54      }
55
56      public int isAdmin() {
57          return admin;
58      }
59
60      public void setAdmin(int admin) {
61          this.admin = admin;
62      }
63
64      public String getRemoteIpAddr() {
65          return remoteIpAddr;
66      }
67
68      public void setRemoteIpAddr(String remoteIpAddr) {
69          this.remoteIpAddr = remoteIpAddr;
70      }
71
72      public String getRemoteHost() {
73          return remoteHost;
74      }
75
76      public void setRemoteHost(String remoteHost) {
77          this.remoteHost = remoteHost;
78      }
79
80      public long getNumVisits() {
81          return numVisits;
82      }
83
84      public void setNumVisits(long numVisits) {
85          this.numVisits = numVisits;
86      }
87
88      public long getFirstVisit() {
89          return firstVisit;
90      }
91
92      public void setFirstVisit(long firstVisit) {
93          this.firstVisit = firstVisit;
94      }
95
96      public String getNewIpAddr() {
97          return newIpAddr;
98      }
99
100     public void setNewIpAddr(String newIpAddr) {
101         this.newIpAddr = newIpAddr;
102     }
103
104
105     public void resetLastVisited(){
106         this.lastVisited = Instant.now().getEpochSecond();
107     }
108
109     public void incrementVisits(){
110         this.todaysVisits = (!isNewDay())? ++this.todaysVisits:1;
111         this.numVisits++;
112     }
113
114
115     private boolean isNewDay(){
116         String pattern = SUCCESS.DATE_FORMAT.toString();
117         SimpleDateFormat simpleDateFormat = new SimpleDateFormat(pattern);
118         if(!simpleDateFormat.format(new Date(this.lastVisited * 1000)).equals(
119                 simpleDateFormat.format(new Date(Instant.now().getEpochSecond() * 1000)))) {
120
121             return true;
122         }
123         return false;
124     }
125
126
127     public int getAdmin() {
128         return admin;
129     }
130
131     @Override
132     public String toString() {
133         return "RemoteHost{" +
134                 "remoteIpAddr='" + remoteIpAddr + '\'' +
135                 ", newIpAddr='" + newIpAddr + '\'' +
136                 ", remoteHost='" + remoteHost + '\'' +
137                 ", numVisits=" + numVisits +
138                 ", todaysVisits=" + todaysVisits +
139                 ", firstVisit=" + firstVisit +
140                 ", lastVisited=" + lastVisited +
141                 ", admin=" + admin +
142                 '}';
143     }
144
145     public static long getSerialversionUID() {
146         return serialversionUID;
147     }
148
149 }
150
```

**********enum Layer**********

ENUM.java

```java
package com.emailManager.enums;


public interface ENUM {
    /**
     * This interface is implemented by both the FAILED enum class and the SUCCESS enum class.
     * Hence, any method 'method(ENUM enum)' is allowed to receive Objects from both classes
     */
}

```

FAILED.java

```java
package com.emailManager.enums;

/**
 * enum class for Error responses
 */
public enum FAILED implements ENUM {
    INVALID_PARAMETERS("Invalid Parameters"),
    MEM_CACHED_ERROR("Memcached failed to connect"),
    EMAIL_ADDRESS_DAILY_MAX_EXCEEDED("Daily max emails from this address reached, please retry after midnight"),
    REMOTE_HOST_DAILY_MAX_EXCEEDED("Daily max emails from this remoteHost reached, please try tomorrow"),
    INVALID_EMAIL_ADDRESS("Invalid email address");

    private final String message;

    FAILED(final String msg) {
        this.message = msg;
    }

    public String toString() { return message; }

}

```

SUCCESS.java

```java
package com.emailManager.enums;

/**
 * enum class for success messages
 */
public enum SUCCESS implements ENUM{
    THANKS("Thanks for contacting me!"),
    MESSAGE_FROM("New email from "),
    MyEmail("sample@gmail.com"),
    MICRO_SERVICE_ON("Ok, this microservice is running!"),
    STATUS("Status: "),
    DATE_FORMAT("yyyy-MM-dd"),
    MESSAGE_BODY("Thanks for your interest in my portofolio, I'm going to read your email as soon as possible!"),
    SECRET_KEY("secretKey1"),
    TOKEN_KEY("secretToken1"),
    TOKEN_HEADER("secretToken2"),
    OK("ok"),
    LINK("<a href=\"http://localhost:8081\">www.rdiaz.com</a>");

    public final java.lang.String message;

    SUCCESS(final java.lang.String msg) {
        this.message = msg;
    }

    public java.lang.String toString() { return message; }


}

```