**<PROJECT 중간보고서>**

**IoT 기술을 활용한 스마트케어 시각화 웹 구현**

**Team E : 유재민, 이재동, 이현재**

**엄현상 교수님**

**윤지수 Master (SK Planet)**

Table of Contents

1. Abstract 2

2. Introduction 3

3. Background Study 3

3.1. 관련 접근방법/기술 장단점 분석 4

3.2. 프로젝트개발환경 5

4. Goal/Problem & Requirements 6

4.1. 프로젝트의 목적 4

4.2. 프로젝트 요구사항 5

5. Approach 7

5.1. 라이브러리 배포 7

5.2. 위험상태 감지 8

6. Project Architecture 7

6.1. Architecture Diagram 7

6.2. Architecture Description 8

7. Implementation Spec 3

7.1. Input/Output Interface 3

7.2. Inter Module Communication Interface 3

7.3. Modules 3

8. Current Status 12

8.1. Study 3

8.2. Code Work 3

9. Future Work 13

10. Division & Assignment of Work 13

11. Schedule 14

# **Abstract**

본 프로젝트는 스마트기기로부터 수집되는 실시간 health data를 입력 받아, 이를 목적에 맞는 스마트케어 전용 차트로 표현하여 웹 페이지에 시각화 하는 것이 목적이다. 이를 수행하기 위해서는 Front-end와 back-end, 그리고 chart library를 구현해야 한다. 단 chart library를 구현하는 과정은 google chart나 chartJS 등의 외부 오픈 소스 라이브러리를 사용하는 것이 아니라, smartcare의 목적에 부합한 library를 직접 개발하도록 한다.

해당 프로젝트의 핵심은 health data를 차트로 만들어주는 라이브러리를 직접 구현하는 것이다. 이미 여러 차트 라이브러리가 상용화 되어 있지만, 스마트케어를 표현하기에 적합한 차트는 찾아보기가 힘들다. 스마트케어와 다른 data의 차이점은 바로 정상범위와 비정상범위가 있으며, 이를 사용자에게 시각화하는 것이 굉장히 중요하다는 점이다. 단순히 신체 정보 값을 그 자체를 그래프로 표현하는 것으로는 사용자에게 큰 임팩트를 줄 수 없다. 해당 정보가 정상 범위에 있는 것인지, 위험한 상태인지에 대한 정보를 시각적으로 표현할 수 있어야 비로소 의미가 있다. 이를 위해 센서값이 위험 범위에 포함될 경우에 chart library 자체에서 경고 표시를 해줌과 더불어 사용자 상태의 파악 하는 것을 시각적으로 도와줄 수 있는 애니메이션 등 또한 추가할 것이다.

Client(Front-end)에서는 스마트케이를 위한 웹사이트를 구현할 것이며 chart library를 사용하여 센서 정보를 차트로 시각화 할 것이다. 다양한 사용자들의 요구를 충족시키기 위해서 다양한 최신 브라우저(Chrome, Firefox, Safari 등)에서 작동 가능하도록 할 것이며, 모바일 환경 등을 지원하기 위해 반응형 웹을 만들 것이다. Client에서 센서값을 받아오기 위해서 Server와 통신을 해야하는데, 이는 HTTP 통신으로 이루어질 것이다. Server에서 미리 파싱된 데이터 값을 json 형태로 받아 이를 시각화한다.

Server(Back-end)에서는 센서에서 전송된 실시간 데이터를 파싱하여 client에 전송하는 일을 한다. Sensor에서 data를 받을때는 MQTT 프로토콜을 이용한다. 이는 server에 설치되어 있는 mosquito(MQTT broker) 모듈이 sensor에서 보낸 데이터를 nodeJS 모듈로 보내주는 과정을 통해 이루어진다. 해당 데이터를 단순히 client에 전송하는 것뿐만 아니라, 추후에 big data 분석이 가능하도록 database를 구축하여 저장한다. 마지막으로 주기적으로 database를 분석하여 사용자의 상태가 위험하다면 주치의와 사용자에서 SMS와 E-mail을 전송하여 상태의 심각함을 알린다. 전체적인 서버는 아마존의 AWS 서버를 사용할 것이며, 대부분의 모듈들은 nodeJS로 구현할 것이다. Database는 mysql을 사용하며, SMS와 E-mail을 보내는 것은 nodeJS 모듈과 외부 SMS 혹은 E-mail 서버가 통신하는 방식으로 구현할 것이다.

# **Introduction**

최근 들어 스마트케어에 대한 관심이 나날이 높아지고 있다. 오래전부터 다양한 신체정보를 감지할 수 있는 센서가 출시되었고, 심지어 근래 들어 출시되는 여러 휴대용 기기에 이를 위한 센서가 포함하고 있다. 세계적으로는 고령화와 건강에 대한 관심 증가로 Heathcare 관련 IoT분야가 크게 성장할 것으로 전망되고 있다. Healthcare 분야의 IoT는 2020년 까지 7억7400만개 정도가 연결될 것으로 예상되고 있다. 이러한 하드웨어의 발전과 더불어 휴대용 기기 제조사에서는 자신들이 만든 센서와 연동되는 어플리케이션을 출시하여, 건강에 관심이 많은 사용자들의 수요를 충족해왔다. 하지만 조금만 살펴보아도 이러한 어플리케이션은 진정한 의미에서의 스마트케어와는 거리가 멀다는 것을 알 수 있다. 실제로 아이폰이나 갤럭시 등에 탑재되어있는 스마트케어 관련 어플리케이션을 사용해보면 단순히 현재 정보와 간략한 통계를 제공 해줄 뿐이다. 따라서 해당 프로젝트에서는 진정한 스마트케어를 가능하게 하는 Front-end와 Back-end를 아우르는 프레임워크를 제시 및 구현하고, 센서 정보를 시각화 해주는 스마트케어 전용 chart library를 구현한다.

해당 프레임워크를 웹 기반으로 설정한 이유는 웹 환경은 핸드폰 어플리케이션에 비해 다양한 환경에서 사용이 가능할 뿐만 아니라, 스마트케어 정보를 웹에서 시각적으로 표현하기 위한 tool이 전무하기 때문이다. 인터넷에 배포되고 있는 오픈소스 웹차트 라이브러리를 통해 스마트케어 데이터를 시각화할 수는 있지만 그런 라이브러리에도 부족한 점이 많다. 신체정보의 특징에 따라 직관적으로 정보를 전달하거나 위험상태 및 정상상태를 구분할 수 없고 스마트케어 정보라는 특수성에 기인하여 차트를 시각화할 수도 없다. 이러한 현 상황에서 본 프로젝트를 통해 만드는 스마트케어 전용 chart library 및 전체적인 프레임워크는 미래의 스마트케어 시장에 유의미한 기여를 할 수 있을 것으로 생각된다.

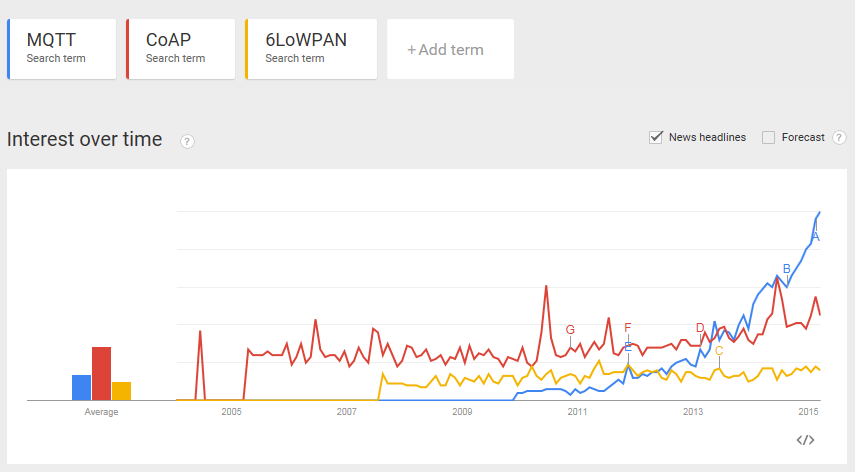
# **Background Study**

**3.1. 관련 접근방법/기술 장단점 분석**

해당 프로젝트를 시작하기 전에 다양한 기술에 대해 공부하고 분석해야 했다. Front-end의 경우 HTML, CSS와 Bootstrap을 이용하여 반응형 웹을 구현하기로 쉽게 결정할 수 있었다. 이에 반해 결정하기가 Back-end, IoT 및 chart library 관련 주제는 세 명 모두 관련 경험이 전무했기 때문에 결정하기가 까다로웠다. IoT의 경우, 이를 구현하기 위한 전체적인 틀에 대한 이해가 필요했으며 특히 어떠한 protocol을 사용해서 구현하는지에 대한 분석이 필요했다. 아래의 항목에서는 각각 IoT, Back-end, chart library를 구현하기 위한 기술에 대한 간단한 분석을 하였다.

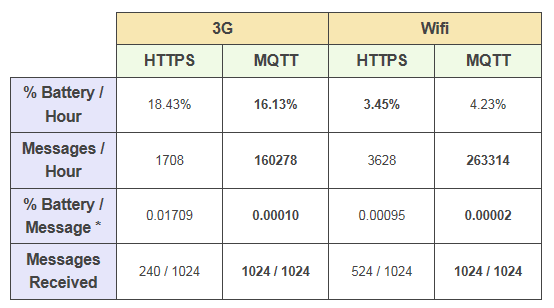
**3.1.1. IoT Protocol**

본 프로젝트에서는 가상의 센서 데이터 값을 서버로 전송하여 이를 관리한다. 이를 위해서는 센서와 서버를 연결하는 IoT Protocol이 필요하다. 다양한 프로토콜 중에 MQTT를 선택하기로 하였는데, publish-subscribe 기반의 light weight message를 사용하여 최근에 가장 널리 쓰이는 프로토콜이기 때문이다.



<그림 3-1> IoT Protocol 비교

가장 처음에는 단순히 HTTP 통신을 이용해서 센서값을 서버로 전송하려고 하였다. 하지만IoT 관련 공부를 하다보니 HTTP는 IoT를 지원하기에 적합하지 않은 프로토콜이라는 것이 대부분의 의견이였다. HTTP Protocol과 MQTT의 가장 큰 차이점은 data 전송 방식에 있다. HTTP는 기본적으로 Request and Response 방식으로, request를 받을 때만 data를 전송한다. 따라서 센서의 upload하는 시간과, 실제로 서버에서 이를 download하는 시간이 다르게 된다. 하지만 MQTT는 publish and subscribe 방식으로, publish된 message는 이를 subscribe하고 있는 모든 client에게 바로 전달되는 장점이 있다. 또한 broker을 통한 many-to-many 통신이 가능하며 HTTP에 비해 저전력으로 전송이 가능하다.



<그림 3-2> HTTP와 MQTT의 비교

**3.1.2. Back-end**

Back-end의 경우에는 asp, nodeJS 혹은 php 중 어떤 것을 선택해서 구현하는 것이 좋을지에 대한 이해가 필요했다. 앞에서 다룬 IoT 관련 이슈와 마찬가지로 프로젝트 팀원 모두 관련 기술에 대한 이해가 전무했으므로 쉽게 결정을 내리기 어려웠다. 결과적으로 nodeJS를 사용하기로 하였는데, 우선 chart library를 만들기로 한 언어인 javascript를 사용하며 최종적으로 구현하기로 한 single-page application에 가장 적합한 언어이기 때문이다.

**3.2. 프로젝트 개발 환경**

# **Goal/Problem & Requirements**

**4.1. 프로젝트의 목적**

현재 시중에서는 많은 차트 라이브러리들이 거래되고 있고 그 중에서는 오픈소스로 공유되고 있는 것도 많다. 그러나 범용성과 개방성을 유지해야 하는 라이브러리의 특성상 차트 라이브러리가 특수한 기능을 가지기는 어렵다. 기능이 많을수록 라이브러리는 사용하고 이해하기에 어려워지기 때문이다. 따라서 시중에 라이브러리는 많지만 특별한 기능을 원하는 사람들이 찾고 있는 것들은 드물다.

헬스케어는 최근 가장 인기 있는 분야 중 하나이다. 대부분의 스마트폰과 스마트워치 등의 기기에 필수적으로 생체정보 센서가 하나 둘씩은 들어있다. 이제 사람들은 운동량, 수면주기, 심박수 등 스스로의 건강에 대한 정보를 스스로 확인하는 것을 당연시하게 되었다. 그러한 데이터들을 단순히 스마트폰 어플리케이션뿐만 아니라 웹사이트에서 확인하고 싶어하는 사람들도 많다. 그러나 그러한 사람들을 위한 차트 라이브러리는 시중에 없다.

따라서 본 프로젝트는 헬스케어를 위해 특화된 라이브러리를 만드는 것을 목표로 한다. 그를 위해 범용성을 다소 희생할 수는 있겠지만 기본적인 차트 라이브러리의 기능은 모두 포함시킬 것이다.

**4.2. 프로젝트 요구사항**

일단 본 프로젝트는 기본적인 차트 라이브러리의 기능을 모두 지원해야 한다. 따라서 가장 핵심적인 차트인 꺾은선, 막대, 원 그래프를 필수적으로 포함해야 한다. 각 차트에 대해서는 유저 취향에 따라 어느 정도의 옵션화가 가능하게 한다. 필수적인 옵션으로는 그래프에 들어갈 데이터의 숫자, 그래프의 테마, 축의 값 표시 여부 등이 있다. 추가적인 옵션으로는 꺾은선 그래프의 interpolation 여부, 막대 그래프의 막대 굵기 등이 있다.

사용자 생체정보는 유저 입력이 아닌 센서 자체로부터 MQTT 프로토콜을 이용하여 받는다. 따라서 전송받은 데이터를 저장하고 클라이언트 측으로 넘겨줄 서버가 필요하다. 해당 서버에서는 데이타베이스(MYSQL)를 이용하여 전송된 사용자 생체정보를 저장하고 JSON 파일을 클라이언트에 넘겨줌으로써 차트에 필요한 데이터를 전달한다. 본 프로젝트는 서버를 구축하는 언어로 NODE.JS를 사용한다.

# **Approach**

**5.1. 차트 라이브러리 배포**

본 프로젝트에서 완성된 차트 라이브러리는 오픈소스로 배포할 것이다. 그러나 본 프로젝트가 서버측 기능과 클라이언트측 기능이 모두 필요한만큼 두 가지 버전으로 배포할 계획이다. 첫 번째 버전은 위험상태 경고기능 및 데이터관리기능 등 서버측 코드가 필요없는 버전으로서 독립적인 차트 라이브러리의 기능을 한다. 두 번째는 모든 서버 코드를 포함한 것으로서 의도된 기능을 수행하기 위해서는 유저가 필요한 NODE.JS 모듈이나 MOSQUITTO BROKER 등을 설치해야 한다.

**5.2. 위험상태 감지**

심박수와 혈압 등을 이용하여 위험상태를 감지하는 알고리즘은 이미 많은 사람에 의해 연구되었다. 그러한 알고리즘 중 본 프로젝트에서 구현하고자 하는 목적에 가장 적합한 알고리즘을 참고하여 새로운 알고리즘을 개발하려 한다. 본 라이브러리가 생체정보를 이용하여 아무리 정확하게 위험상태를 감지한다 하더라도 센서 자체의 오류가능성을 고려한다면 확실한 정보가 될 수 없다. 따라서 본 알고리즘은 신빙성 있는 출처에서 찾은 것이되 가장 보수적인 것으로 선택한다. 따라서 의미 없는 데이터에 의해서 이상이 없는데도 알람이 울리는 경우를 최대한 방지한다.

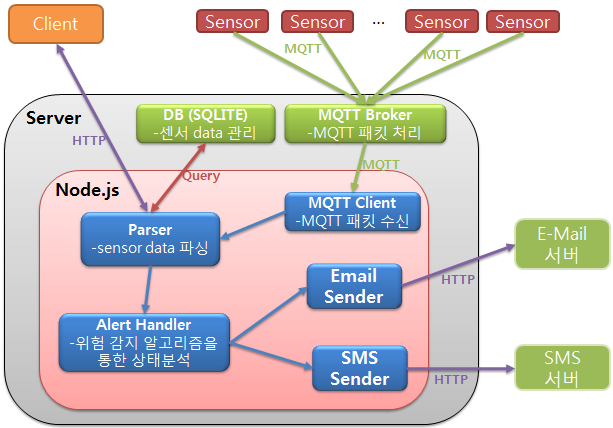
각각의 위험 상태 감지 알고리즘은 다음과 같다. 우선 체온의 경우 데이터베이스에서 매 10분마다 최근 10분 간의 데이터를 수집한 뒤, 평균을 낸다. 해당 값이 36도 이하이거나 37.5도 이상일 경우 경고를 한다. 수면의 경우 두가지 권고 알고리즘으로 구성되어 있다. 첫번째의 경우 데이터베이스에서 최근 3일간의 데이터를 수집한 뒤 총 수면시간을 계산한다. 계산된 수면 시간이 18시간 이하일 경우 수면 권고 문자를 보낸다. 두번째의 경우 새벽 1시 이후부터 매 시간마다 최근 1시간의 데이터를 수집한 뒤 수면시간을 계산한다. 최근 1시간 동안의 수면 시간이 30분 이하일 경우 수면 권고 문자를 보낸다. 마지막으로 심박수 또한 두가지로 알고리즘이 나뉜다. 이는 젊은 사람들의 경우 낮은 심박수가 크게 위험이 되지 않지만 나이가 든 사람의 경우 저혈압이 큰 문제가 되기 때문이다. 따라서 데이터베이스에서 매 10분마다 최근 10분 간의 데이터를 수집한 후 평균을 낸 뒤, 나이가 40세 미만인 경우 분당 심박 수가 40이하이거나 100 이상인 경우 경고를 한다. 다만 나이가 40세 이상인 경우 분당 심박수가 60이하이거나 100 이상인 경우 경고 문자를 전송한다.

# **Project Architecture**

**6.1. Architecture Diagram**

<그림 6>은 본 프로젝트의 전체구조를 대략적으로 나타낸 것이다. 본 프로젝트의 결과물은 크게 클라이언트측 코드, 서버측 코드, 기타로 나뉜다. 서버측 코드는 NODE.JS로 짜여있으며 MQTT 프로토콜을 이용하여 센서와 통신하는 것, 위험상태를 감지하여 SMS와 메일로 경고하는 것, 센서에게 전송받은 데이터를 데이터베이스로 관리하는 것, JSON 파일을 이용하여 데이터를 클라이언트와 공유하는 것 등의 일을 한다. 클라이언트측 코드에서는 HTML 파일과 JS 파일을 이용하여 사용자의 웹 브라우저에 차트를 보여주는 일을 한다. 기타에는 메일을 전송해주는 구글 서버, SMS를 전송해주는 CAFE24 서버, 실제 데이터를 측정하는 센서들이 포함된다.

**6.2. Architecture Description**

****

**<그림 6>**

**6.2.1. 클라이언트**

클라이언트 코드에는 INDEX.HTML과 JCHARTS.JS가 포함된다. INDEX.HTML은 웹페이지를 구성하는 HTML 파일이다. 실제로 배포될 때는 라이브러리에 포함되지 않겠지만 라이브러리의 기능을 점검하고 시연하기 위해서는 꼭 필요하다. INDEX.HTML에서 JCHARTS.JS를 불러와서 차트를 그린다. JCHARTS.JS는 실제 차트를 그리는 기능을 모두 포함하고 있다. 서버에 존재하는 JSON 파일을 불러와 파싱하여 CANVAS를 이용해 실제 그래프를 그려준다. HTML 파일을 통해 사용해야 하기 때문에 전체 파일이 HTML DOCUMENT의 WINDOW를 인자로 받는 함수로 이루어져 있다.

**6.2.2. 서버**

서버 코드는 MQTT.JS, SMS.JS, MAIL.JS, ALERT.JS로 이루어져 있다. MQTT.JS는 센서들로부터 데이터를 입력받고 그것을 저장하는 역할을 한다. 전체 데이터는 데이터를 받은 시각과 함께 MYSQL에 저장한다. 그리고 실제 그래프를 그리기 위해 필요한 부분적인 데이터는 JSON 파일을 통해 클라이언트에 넘겨준다. ALERT.JS 파일은 MQTT.JS 파일이 만든 데이타베이스를 사용하긴 하지만 MQTT.JS 파일과는 독립적으로 동작한다. 위험상태를 감지하기 위해 데이터베이스에서 최근 몇 분 ~ 몇 시간의 데이터를 읽어온 다음 사용자가 현재 위험상태에 처해 있는지 아닌지를 판단한다. 만약 위험상태에 처해있을 경우 SMS.JS와 MAIL.JS에 각각 신호를 보내어 SMS와 메일로 사용자에게 경고한다. SMS.JS와 MAIL.JS는 각각 SMS와 메일 전송을 담당하는 간단한 모듈이다.

**6.2.3. 기타 필요한 것들**

기타 아키텍쳐에 포함된 것들로는 센서, SMS서버, 메일서버, 웹브라우저가 있다. 본 프로젝트는 유저입력이 아닌 센서데이터를 사용하기 때문에 센서가 필수적으로 필요하다. MQTT 프로토콜을 사용하기만 한다면 어떤 센서도 본 프로젝트를 위해 활용될 수 있다. 현재는 센서장비를 구하지 못했기 때문에 다른 서버에서 SHELL 파일을 이용해 보내는 가짜 데이터를 테스트에 사용한다. SMS서버는 CAFE24 호스팅을 통해 얻은 것이다. 해당 서버에 수신자, 발신자, SMS내용 등을 담아 HTTP POST REQUEST를 보내면 해당 서버가 대신 SMS메시지를 보내준다. 현재 메일은 GMAIL 계정을 통해 보내고 있기 때문에 메일서버는 구글서버이다. 기본적으로 범용성을 제공해야 하는 라이브러리이기 때문에 다양한 웹브라우저에 대해 차트가 잘 표현되어야 한다.

# **Implementation Spec**

**7.1. Input/Output Interface**

**7.1.1. Sensor Data**

다양한 센서로부터 MQTT 프로토콜을 이용하여 데이터를 넘겨받는다. 같은 프로토콜을 사용하기만 하면 센서의 종류나 개수에는 제한이 없다. 단지 PUBLISH하고 SUBSCRIBE하는 주제와 데이터의 형식만 같으면 된다. 서버가 MQTT 프로토콜을 이용하여 데이터를 넘겨받으면 그것을 알아서 클라이언트에 넘겨주고 클라이언트는 그 데이터를 이용하여 차트를 그린다.

**7.1.2. User Customization**

기본적으로 라이브러리이기 때문에 사용할 때 여러 가지 옵션을 줄 수 있다. 옵션은 크게 두 가지 방식으로 줄 수 있다. 하나는 CONFIG.JSON 파일을 이용하여 서버 측 기능에 대한 옵션을 제공하는 것이고 다른 하나는 HTML 파일 안에서 차트를 그릴 때 옵션으로 여러 가지 인자를 제공하는 것이다. CONFIG.JSON 파일에서는 위험상태감지 알고리즘의 사용여부나 변형된 알고리즘의 사용여부, SMS의 전송여부, 메일의 전송여부 등을 설정할 수 있다. HTML 파일에서는 차트의 종류(막대, 꺽은 선, 원형), 차트의 색(단색 or 그라데이션), 그래프의 모양(smooth or step), 축의 값(x,y 축) 표시, data의 범위, 그래프의 크기 등을 설정할 수 있다.

**7.1.3. Database**

센서로부터 받은 데이터를 시간순서로 잘 정리해야 위험상태를 성공적으로 감지할 수 있다. 따라서 본 프로젝트에서는 MYSQL 데이터베이스를 이용하여 모든 데이터를 데이터 타입, 데이터 값, 데이터를 받은 시간에 따라서 저장한다. 라이브러리를 통해서가 아니라도 해당 데이터베이스에는 MYSQL을 이용하여 접근할 수 있다.

**7.1.4. Browser**

차트는 웹 브라우저를 통해 확인할 수 있다. 최소한 CHROME, FIREFOX, SAFARI의 세 가지 브라우저에서 정상적으로 작동하게끔 만드는 것이 목표이다. INTERNET IXPLORER에서의 동작 여부는 현재 선택사항이다. 브라우저의 창 크기에 따라 반응형으로 차트의 배치나 레이아웃이 달라지게 하는 기능 또한 현재는 선택사항이다.

**7.2. Inter-module Communication Interface**

**7.2.1. HTTP Communication**

본 프로젝트에서는 대부분의 통신을 HTTP로 한다. 기본적으로 브라우저와 HTML 파일의 통신, SMS서버와 라이브러리서버의 통신, 클라이언트 파일과 서버 데이터파일의 통신이 HTTP로 이루어진다. 대부분은 POST와 GET REQUEST를 많이 사용한다.

**7.2.2. MQTT Protocol**

센서와 서버의 통신은 MQTT 프로토콜로 이루어진다. MQTT 프로토콜은 IOT DEVICE에 최적화되어 저전력, 저안정성으로 동작하게끔 디자인된 프로토콜이다. MQTT 프로토콜은 센서가 특정 주제에 대해 데이터를 발행(PUBLISH)하면 서버가 그것을 구독(SUBSCRIBE)하는 방식으로 이루어진다.

**7.2.3. Module Requirement**

굳이 통신할 필요 없는 모듈끼리는 직접적인 코드 의존의 관계로 묶여 있다. HTML 파일과 클라이언트 JAVASCRIPT 파일은 HTML 파일이 JAVASCRIPT 파일을 의존하면서 연결되어 있다. NODE.JS로 작성된 서버측 파일들 또한 서로를 의존하면서 연결되어 있다.

**7.3. Modules**

1. **SERVER SIDE**
   1. **MQTT.JS** : communicates with MQTT and MYSQL.
      1. function push(data, config, type, elem) : pushes a element into an array of a JSON object. Data and type specifies the array. Config is a config object which defines maximum length of the array.
      2. function writeData(data) : writes data into ‘data.json’ file which connects server side and client side.
      3. function getTime : returns current time of Korea by the form of 'YYYY/MM/DD hh:mm:ss.'
   2. **ALERT.JS** : examines the data from MYSQL and alerts people if there is a serious problem. It requires MAIL.JS and SMS.js.
      1. function getData(time) : gets data from the MYSQL. It selects recent data to judge whether it is dangerous or not.
      2. function average(type) : get the average of data depends on its data type. It takes data type as an argument because each data type has a different policy of time interval.
      3. function determineTemp() : determines whether sending alert to user about body temperature.
      4. function determineSleep1() : determines whether sending alert to user about total amount of sleep for last three days.
      5. function determineSleep2() : determines whether sending alert to user about not sleeping after 1AM.
      6. function determineHeart1() : determines whether sending alert to user about heart rate of people younger than 40 years old.
      7. function determineHeart2() : determines whether sending alert to user about heart rate of people older than 40 years old.
      8. function alert(type) : alerts the status to the users. It uses SMS and mailing services together.
   3. **SMS.JS** : is a NODE.JS module which can send SMS to the users.
      1. function encodeBase64(str) : encodes the string and transfers it into base 64 buffer to use CAFE24-SMS module.
      2. function sendSMS(callback) : sends SMS request to CAFE24 server. Then it is sent to the users. Callback is a callback function which is called when message is successfully sent.
   4. **MAIL.JS** : is a NODE.JS module which can send mails to the users.
      1. function sendMail(data, callback): sends a mail to the user.
   5. **PACKAGE.JSON** : defines NPM dependencies.
   6. **NODE\_MODULES** : is a directory which contains NODE.JS librarires
2. **CLIENT SIDE**
   1. **JCHARTS.JS** : is a main library code file.
      1. function getConfig() : gets configs from CONFIG.JSON file. It is not being used now.
      2. function getData() : gets data from DATA.JSON file. The data are used for drawing charts.
      3. function parseAttr(elem, attr) : parses specific attributes from the element. Each element is an object in a HTML file.
      4. function getXInterval(elem\_number) : calculates intervals between dots in a graph. It is used for line and bar chart.
      5. function getXForIndex(idx, elem\_number) : returns X value of the element from its element number. It uses getXInterval(elem\_number) to calculate the result.
      6. function getYForValue(val, range) : returns Y value of the data value between the range. Range defines minimum and maximum values in the chart.
      7. function drawAxis() : draws axises for the chart. It is used for line and bar charts.
      8. function drawDotLine() : draws a dot line.
      9. function sumSet(set) : returns sum of all values in the set.
      10. function setColorType(grad, section, color) : defines a theme of the chart by 3 parameters.
      11. function renderPieChart() : renders a pie chart.
      12. function setMinMax(set, range) : sets all values in the set to a specific range. Range contains minimum and maximum values.
      13. function renderLineChart(set) : renders a line chart from data of the set.
      14. function renderBarChart(set) : renders a bar chart from data of the set.
      15. function init(elem) : initiates a process drawing a chart for each element from the HTML file.
      16. object Jcharts : is an API which can be used in HTML files.
      17. function render(elems) : renders a chart for the elements. it calls init(elem) for each element.
      18. function update() : is not used now.
   2. **DATA.JSON** : is made from MQTT.JS file. It contains all chart data which can be used for drawing charts in JCHARTS.JS file.
   3. **CONFIG.JSON** : defines several options for server side.
   4. **TEST.HTML** : is used for testing.

# **Current Status**

**8.1. Study**

본 프로젝트를 진행하는 조원 세 명 모두 웹개발 경험이 전혀 없기 때문에 처음에는 웹에 대해 공부하는 것으로 시작했다. 먼저 HTML과 CSS를 공부하며 웹페이지를 만드는 데 익숙해진 다음 JAVASCRIPT 문법과 코딩 스타일을 익혔다. 그 다음에는 그래프를 그리는 데 필요한 HTML5의 CANVAS 라이브러리를 사용하는 방법을 공부했다. MQTT 프로토콜을 이용하여 센서와 통신해야 하므로 MQTT 프로토콜, MOSQUITTO, MQTT 브로커 연결 방법 등에 대해서도 공부해야 했고 기본적인 서버를 구축하기 위해 AWS 사용방법, NODE.JS를 이용한 서버구축 등을 익혔다.

**8.2. Code Work**

현재 기본적인 클라이언트 및 서버의 구조가 잡힌 상태이다. 먼저 서버 측에서 가장 필수적인 모듈이며 센서와의 통신, 클라이언트로의 데이터 전달, MYSQL에의 데이터 저장을 맡는 MQTT.JS의 기본적인 기능이 완성되었다. 그 다음엔 위험상태 경고기능을 맡는 ALERT.JS와 사용자에게 경고해줄 때 필요한 SMS.JS, MAIL.JS가 각각 완성되었다. 클라이언트 측에서는 기본적인 BOOTSTRAP을 이용한 반응형 웹페이지가 만들어졌고 그래프 작성을 담당하는 JCHARTS.JS가 오류 없이 기본적인 세 가지 차트를 그릴 수 있게 되었다.

# **Future Work**

먼저 기본적인 형태로만 완성된 차트 라이브러리를 헬스케어에 특성화된 것으로 개선해야 한다. 심박수, 수면주기, 혈압에 대해 최적화된 디자인과 레이아웃을 찾아낸 다음 가장 직관적으로 정보를 전달할 수 있도록 옵션을 설정해야 한다. 그러면서도 유저의 취향에 맞게 몇 가지 디자인 선택지를 줄 수 있어야 한다. 그리고 본 프로젝트의 핵심적인 기능인 위험감지에 대해서도 알고리즘을 대폭 수정하여 더욱 정확하고 의미있는 경고기능이 되도록 한다. 본 프로젝트의 결과는 오픈소스로 배포하는 것이 목적이기 때문에 코드의 기능이 완성되었다고 하더라도 누구나 쉽게 이해하고 사용할 수 있도록 코드를 모듈화하고 개선하는 것도 빼놓지 않아야 한다. 차후 데이터부하가 심해질 경우 차트 그리기가 느려질 수 있기 때문에 MQTT 통신과 모듈간의 데이터 전달 과정을 최적화하여 서버부하를 최소로 만들어야 한다.

# **Division & Assignment of Work**

|  |  |  |
| --- | --- | --- |
| **분류** | **항목** | **담당자** |
| **Server** | 경고 전송 기능 | 유재민 |
|  | 프로토콜 변환 기능 | 유재민 |
| **IoT** | MQTT 보완 | 이현재 |
|  | 데모 shell 파일 작성 | 이현재 |
| **Web** | Web 디자인 | 이재동 |
|  | 반응형 웹 구축 | 이재동 |
| **Library** | 심박수 chart | 유재민 |
|  | 체온 chart | 이현재 |
|  | 수면주기 chart | 이재동 |
|  | Chart 애니메이션 | 유재민 |
|  | Chart 반응형 기능 추가 | 이현재 |
|  | 위험 감지 알고리즘 | 이재동 |

# **Schedule**

