**Design Specification**

**Project Weer station**

**Groep B**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Functie** | **Naam** | **Paraaf** | **Datum** |
| **Team** | Projectleider | Lucas van Gastel |  |  |
| **Goedkeuring** | Opdrachtgever |  |  |  |
| **Goedkeuring** | Opdrachtgever |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Versie | Datum | Naam | Beschrijving |
| 0.1 | 17-09-2020 | Lucas van Gastel | Opzet ontwerpdocument |
| 0.2 | 17-09-2020 | Lucas van Gastel | Generieke informatie ingevuld |
| 0.3 | 17-09-2020 | Lucas van Gastel | Setup for chapter 2,3 and 4. |
| 0.4 | 17-09-2020 | Lucas van Gastel | Software layout of embedded system |

**Inhoudsopgave**

[1. Introduction 4](#_Toc52997341)

[1.1. Background information 4](#_Toc52997342)

[1.2. Legal state of the document. 4](#_Toc52997343)

[1.3. References 4](#_Toc52997344)

[1.4. Subject language 4](#_Toc52997345)

[2. System outline 6](#_Toc52997346)

[2.1. Context diagram - system 6](#_Toc52997347)

[2.2. State diagrams 7](#_Toc52997348)

[2.2.1. QT – The desktop application 7](#_Toc52997349)

[2.2.2. NodeMCU-32s – The embedded system 7](#_Toc52997350)

[2.3. Activity diagram 8](#_Toc52997351)

[2.3.1. QT – The desktop application 8](#_Toc52997352)

[2.3.2. NodeMCU-32s – The embedded system 8](#_Toc52997353)

[3. Hardware-Software Interface 9](#_Toc52997354)

[3.1. Context diagram - NodeMCU-32s 9](#_Toc52997355)

[4. Software design 10](#_Toc52997356)

[4.1. General outline 10](#_Toc52997357)

[4.1.1. QT – The desktop application 10](#_Toc52997358)

[4.1.2. NodeMCU-32s – The embedded system 10](#_Toc52997359)

[4.2. Module delineation 11](#_Toc52997360)

[4.2.1. Wifi 11](#_Toc52997361)

[4.2.2. HTTPClient 13](#_Toc52997362)

[4.2.3. Data 15](#_Toc52997363)

[4.2.4. Module X 16](#_Toc52997364)

[4.2.5. Module X 16](#_Toc52997365)

[4.2.6. Module X 17](#_Toc52997366)

# Introduction

This document contains the specific design choices and specifications of the project. Whereby it primarily lies its focus on the software aspect, it is not limited to such. In some cases, links will be established between hardware and software.

The general system layout and design is explained in chapter 2. Communication between hardware and software is treated in chapter 3, and last of all, chapter 4 contains the software designs.

## Background information

The goal of this project is to create an autonomous weather station. This weather station is required to establish a wireless connection to an online environment where data can be managed and stored.

The project is split into two main programs. One running on a Windows machine, which has as main point of focus of visualizing information gotten by the microcontroller. The second is an autonomous microcontroller which gathers multiple sensor readings and uploads these to a database.

## Legal state of the document.

This document is written by students under the authority and responsibility of Avans Hogeschool. All conclusions and or recommendations followed by a third party is done so at own risk.

## References

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Ref | Filename | Titel | Auteur | Versie |
| [1] | Bijv. URS | Bijv. User Requirement Specification |  |  |
| [2] | Bijv. FS | Bijv. Funcional Specifications |  |  |

## Subject language

|  |  |  |
| --- | --- | --- |
| Afkorting | Term | Omschrijving |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

# System outline

This chapter describes the general system layout and interaction. Since the project consists of two major lines of work, an application and an embedded system. The system layouts will be split between these divisions. The layout will, however, explain the manner in which these two systems interact to form one entity.

The first sub-chapter will elucidate into the embedded system and how this interacts with the environment. Followed by the state diagrams of both the embedded system and desktop application. Finally, the activity diagrams will look to bring greater clarification into the workings of the systems from an outside perspective.

## Context diagram - system

*???*

## State diagrams

The state diagrams aim to provide a general insight into the internal workings. This chapter has been split into two main parts, the QT GUI and embedded system, since these are two separate applications.

The connectivity between the two systems is done via an external server-side script. This script is used to post data towards, splits it into the correct columns and queries it into the database. The QT application proceeds to poll the database in order to receive the data.

## QT – The desktop application

The desktop application follows the following pattern:

1. The window startup.
2. The initialization.
3. Try to connect to the database.
4. If the connection to the database is successful the application goes on to making and displaying the graphs.
5. If the connection is not successful the application ask the user if he want to reconnect.
6. If the user wants to reconnect the application goes back to step 3. Try to connect to the database.
7. If the user does not want to reconnect the application will go on.
8. Creating and displaying graphs.
9. Waiting for user to close the application.

## NodeMCU-32s – The embedded system

The embedded system follows a simple pattern, it initializes, reads the sensory data, posts it to a server, and proceeds to go into sleep mode for a minute before restarting the cycle.

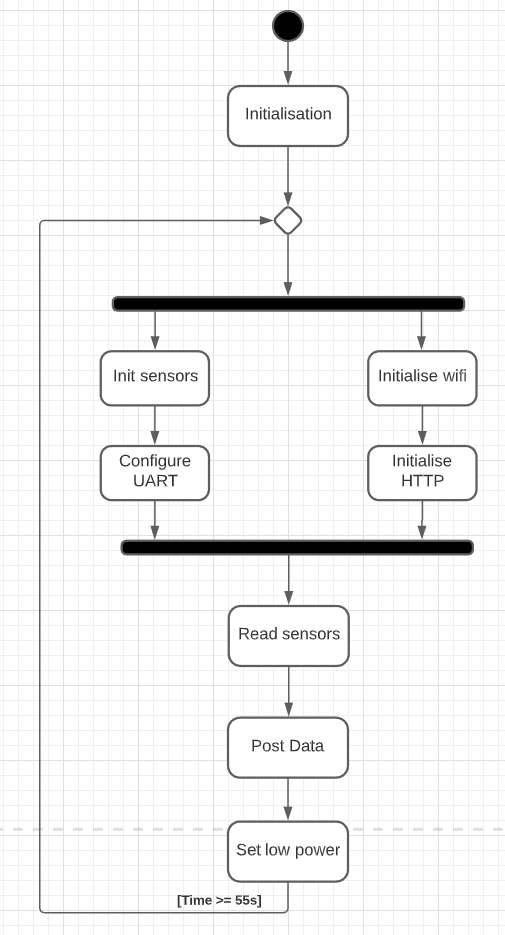
## Activity diagrams

The activity diagrams are meant to give a slightly more in-depth view into the main system, hereby it will gloss over the internal works of added workings such as the Wifi and HTTPClient wrappers to preserve simplicity as these will be explained in their respective chapters.

## QT – The desktop application

The application will start up by creating the application and the mainWindow, then the initialization, making a connection with the database, if the connection is successful the data will be retrieved from the database, if the connection is not successful the user will be asked what to do next. Th user will have the choice to try to reconnect or not to try to reconnect. Then the graphs will be made and displayed and after that the application will be waiting for the user to close the application.

## NodeMCU-32s – The embedded system

The embedded system will initialize itself, followed by simultaneously setup sensors and Wi-Fi, followed by posting the data into the database. After this it will enter a low power mode to limit battery usage, as a result the Wi-Fi connection will be lost, this will be restored after the system has been woken up by an internal interrupt of the RTC clock.

# Hardware-Software Interface

This chapter will expound on the interactions with the environment. Whereby the sensors and general hardware of the NodeMCU-32s will be explained in further detail.

## Context diagram - NodeMCU-32s

???

# Software design

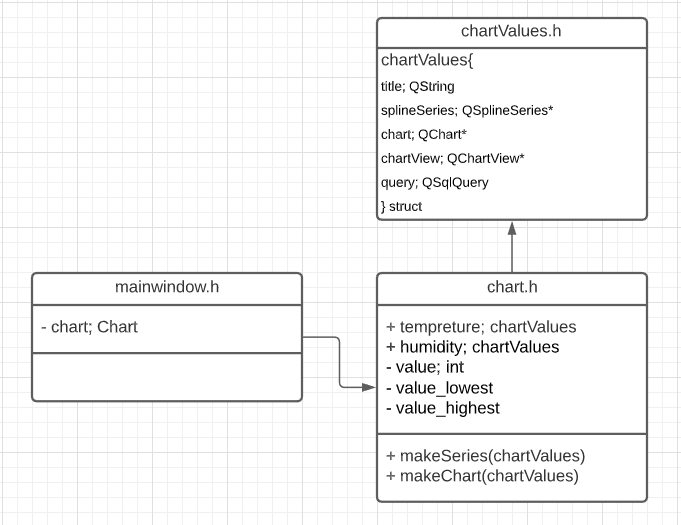
Entities in the software are split up into a two-dimensional lay-out, hereby the vertical lane will indicate the execution order of the system. The horizontal lane expands on methods called concurrently or at different stages in the system.

## General outline

This chapter glosses about the general layout of scripts and their purposes.

## QT – The desktop application

|  |  |  |
| --- | --- | --- |
| .h files | .cpp files | Doel |
|  | main.cpp | Main startup of the system. |
| mainwindow.h | mainwindow.cpp | Creating the main window |
| Chart.h | Chart.cpp | Making a connection with the database, getting data from the database, making graphs |
| chartValues.h |  | Struct containing data that is necessary for creating graphs |
|  |  |  |



## NodeMCU-32s – The embedded system

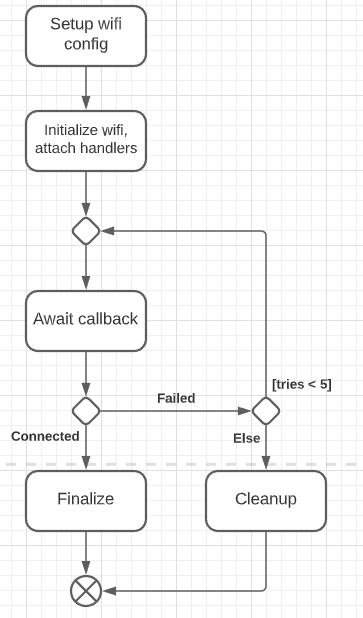
|  |  |  |
| --- | --- | --- |
| .h files | .cpp files | Doel |
|  | main.cpp | Main startup of the system. |
| Wifi.h | Wifi.cpp | The Wi-Fi module aims to allow for Wi-Fi connections to be established as an AT module. |
| HTTPClient.h | HTTPClient.cpp | The HTTPClient aims to give an easy implementation to interact with the database. |
| Data.h |  | Struct containing data for HTTP post. |
|  |  |  |
|  |  |  |

## Module delineation

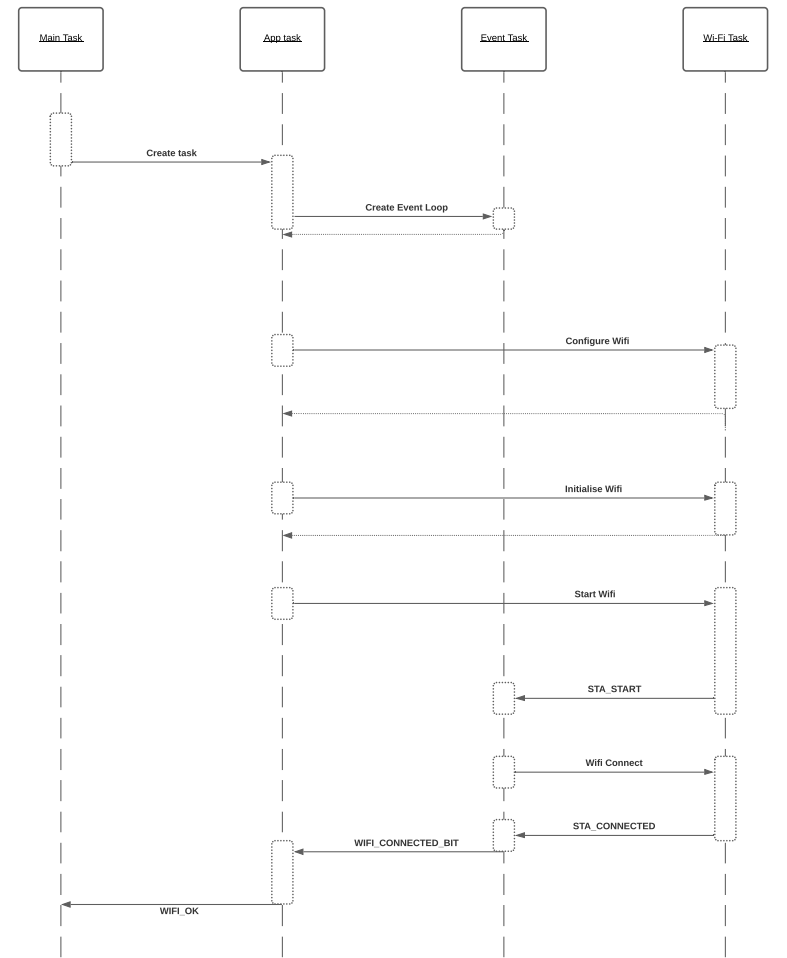
This covers the internal workings of the modules and describes the internal ownership or knowledge of other classes where applicable. However, the primary goal is to expound on the flow and layouts of data in the described modules.

## Wifi

The Wifi module aims to establish a connection to an access point. This is established through the included FreeRTOS libraries: string, freertos/FreeRTOS, freertos/event\_groups, nvs\_flash, esp\_wifi. These libraries are used to initialize the WIFI SSID and password, use multiple tasks, gather all responses of tasks, initialize the NVS in order to preserve a connection and initialize the wifi respectively. This module is a wrapper around the underlying provided methods of the OS.

The general flow of the module is as follows: It configures the connectivity parameters of a Wi-Fi access point, initializes the Wi-Fi and attaches the event handlers used to handle the callbacks. If it fails, the module will attempt five more times, at that point whether it succeeded or not. The module will be cleaned up and, handlers are removed, and resources are freed.

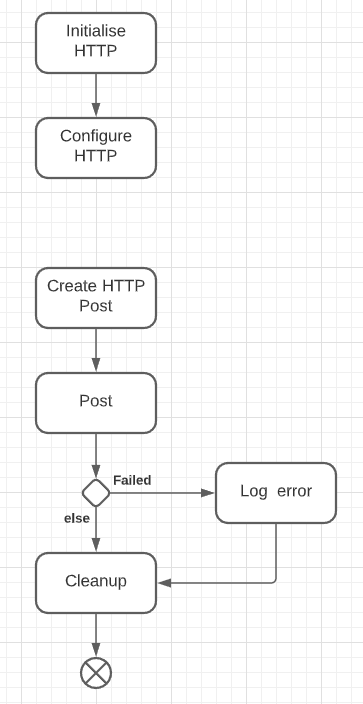
The tasks created are shown in the diagram on the next page. This also shows the flow of functions and callbacks in order.



## HTTPClient

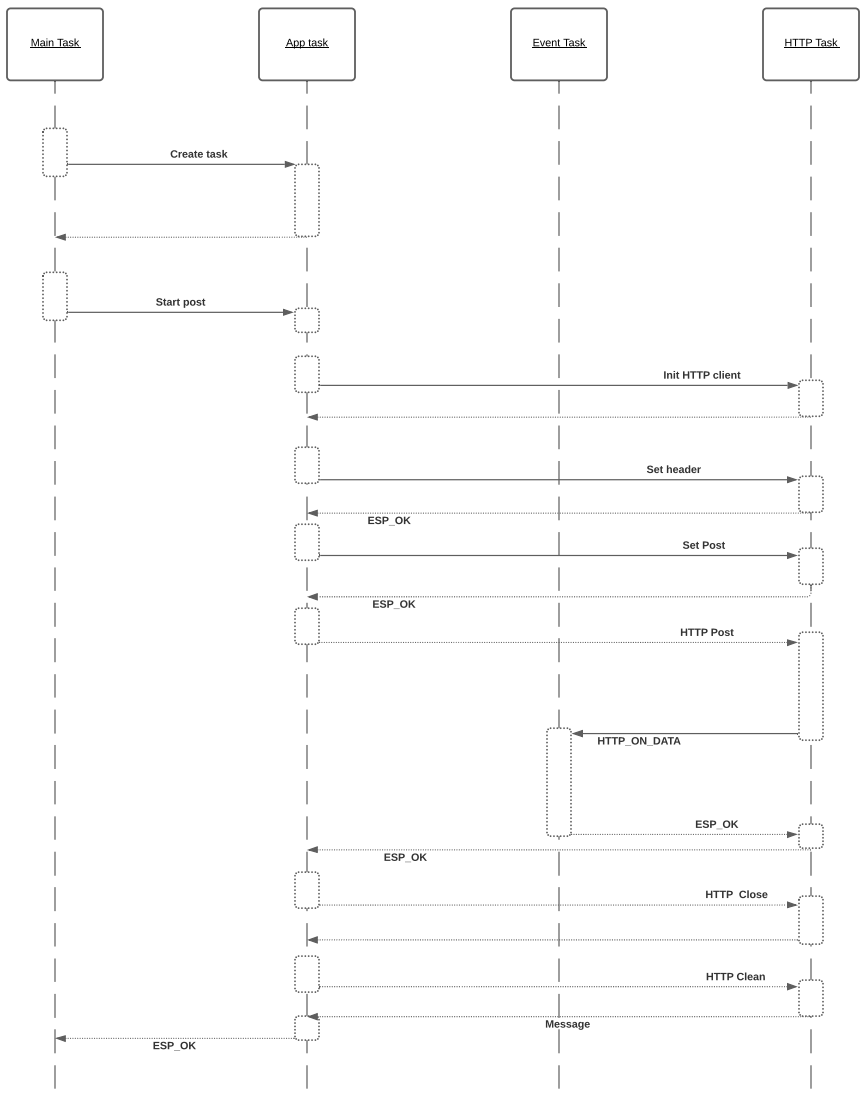
The HTTPClient aims to provide a streamlines wrapper to interact with a server. In order to achieve this, it makes use of a data struct found in Data.h, string and the following FreeRTOS library “esp\_http\_client”

These libraries are used to initialize the server and JSON like post and use the HTTP client interface respectively. This module is a wrapper around the underlying provided methods of the OS.

The class must be initialized, from this point on it will prepare the connection by initializing HTTP and configuring the connectivity strings.

In order to use the HTTPClient, a Data class instance must be created and given to the post method, this will convert it into a JSON object and perform the post request. Following this, the class will be cleaned and prepared for a follow-up post.

The tasks created are shown in the diagram on the next page. This also shows the flow of functions and callbacks in order. The callbacks are used for debug only at this point. For deployment, these will be used for error handling during the post.



## Data

The data class is a container struct, used for storing sensory data and allows for easy conversion into a JSON like object for follow-up post requests. It has one method, the “GetPost” method and the public members which are to be send to a database.

## Module X

## Module X

## Module X