**Report 2**

Eric Kreider

Department of Computer Science and Informatics, Indiana University of South Bend

System Implementation

Professor Shawn Dai

May 2nd, 2022

Table of Contents

Interaction Diagrams - 2

Project Coordination and Progress report - 3

Plan of work - 6

Gantz Chart - 7

Class diagram - 9

Data types – 10

Traceability Matrix - 13

Architectural style – 15

Identifying Sub Systems – 16

Mapping Subsystems to Hardware – 17

Persistent Data Storage – 18

Network Protocols – 19

Global Control Flow – 20

Algorithms and Data Structures - 23

Recent Design Changes – 25

Testing Overview and Methodology – 27

Project Management and Plan of Work - 29

Interaction Diagrams

Below are the interaction diagrams of three fully-dressed use cases referenced earlier in the report. The use cases utilize a number of important design principle that are meant to support the success of the project. One mantra throughout the project is “Don’t Repeat Yourself”. Creating reusable components for the GUIs, functions for frequent blocks of code, and the use of Bootstrap CSS for simple styling helps keep the project focused on the more important aspects of the system (Don’t Repeat Yourself, 2022). Additionally, the system design focused on implementing a minimum viable product, meaning that the project was developed in such a way as to try to create immediate, tangible, functional software that can be iteratively improved upon (Minimum viable product, 2022). This allows for faster deliverables and gives more room for usability testing and additional time to incorporate user feedback. Overall, these design methodologies result in a responsive, customer-focused development environment. For example, in all below listed use cases, minimum viable product was emphasized by developing simple, but functional prototypical versions of the web application pages. After that, the “don’t repeat yourself” methodology was utilized by creating style sheets that apply to all of the elements within a webpage and automatically apply certain screen size controls via added Bootstrap libraries.

Diagram

Description automatically generated

Diagram

Description automatically generated

Diagram

Description automatically generated

Project Coordination and Progress Report

Currently, a good number of use cases have been implemented. The development process followed a back-to-front development that focuses on implementing data flows and serving those dataflows to the webpage. After that, the data will be formatted into an attractive display for end users. Some of this will happen concurrently, but the primary interest is developing the functions that deliver the data before time can be fully allocated to the design portion of the project. The use cases that were outlined to be implemented are as follows:

* View Current Sensor data
* Look at sensor data over time
* Create a new user
* Reset user password
* Change own password
* View sensor module data
* Look at all current sensor data
* Create comments
* Log in
* Export data

Currently, the use cases that need to be implemented still are ‘View sensor module data’, ‘Create a New User’, and the visualization portion of ‘Look at sensor data over time’. These remaining use cases and their associated functions will be the focus of the near future. The code will be viewed, optimized, and tested.

Plan of Work

The original plan of work is referenced below. Based off of the plan of work, the remaining tasks are building webpages, style implementation, usability studies and testing, and finally cleaning and stretch goals. The following Gantz chart has been generated to demonstrate the flow of work for the remainder of the project.

Table

Description automatically generated

Chart

Description automatically generatedGantz Chart

Interaction Diagrams

Class Diagram

Diagram

Description automatically generated

Data Types and Operation Signatures

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | |  |  | | --- | --- | | Class Name: | Sensor | | Attributes: | Hostname(varchar(45)) – Hostname in OS SensorName(varchar(45)) – human readable name  Description (varchar(45)) – Sensor description  ID (int) – unique, autoincrementing identifier  DateAdded(datetime) – date system was added | | Functions: | WriteRecord() – a function that runs off of the data collecting Raspberry Pi. It inserts a record of temperature, humidity, timestamp, and the hostname into the database | | Description: | A sensor is the class for the real world data collecting Raspberry Pi. It contains information about the system, it’s hostname, its human readable name, its description, and its unique ID. It has the ability to add data to the master database for the webpage to query. | |
|  | |  |  | | --- | --- | | Class Name: | SensorRecord | | Attributes: | recordID(int(11)) – auto incrementing record ID  temperature(decimal(10,0)) – the temperature recorded  humidity(decimal(10,0)) – the humidity recorded  sensorID(int(11)) – the ID of the sensor that submitted the record  dateTime(datetime) – the timestamp of the record | | Functions: |  | | Description: | This class represents the individual records that will be manipulated by the webpage, the sensors, and the users. The records are created by the sensor class and are queried by the different users. | |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | |  |  | | --- | --- | | Class Name: | Comment | | Attributes: | ID(int(11)) – auto incrementing record ID  Username(varchar(20)) – Username of commenter  Comment(varchar(120)) – comment made by an individual | | Functions: |  | | Description: | A comment refers to something posted by a user. All users can create comments and will generate a record everyone can see. | |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | |  |  | | --- | --- | | Class Name: | WebUser | | Attributes: | UserName(varchar(20)) – a uniquely identifiable username that identifies end users  UserPassword(varchar(45)) – a password used to authenticate into the system  FirstName(varchar(45)) – The first name of the user  LastName(varchar(45)) – The last name of the user  AccessLevel(int(11)) – the access level of the user. It corresponds to different user groups  Notes(varchar(45)) – A notes area for relevant info  CreationDate(datetime) – the timestamp of the creation of the account | | Functions: | Login(Username, UserPassword) – authenticates the user into the system if they have a relevant account. Creates a session variable  Logout() – clears the logged in session variable and returns the user to the main page  Comment() – Creates a comment on the webpage that is viewable by everyone  ChangePasswordSelf() - | | Description: | A comment refers to something posted by a user. All users can create comments and will generate a record everyone can see. | |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | |  |  | | --- | --- | | Class Name: | QualManager (specialization of WebUser) | | Attributes: |  | | Functions: | ViewTrendData() – views trend data for a specific sensor over a specific time. | | Description: | This class is a specialization of the WebUser class. As such, it inherits all the attributes and functions of the WebUser class. | |
|  | |  |  | | --- | --- | | Class Name: | AppSupport (specialization of WebUser) | | Attributes: |  | | Functions: | ResetOtherPassword() – reset another user’s password if they are locked out of the system  RunDiagnostic() – test if the database is reachable, view last timestamp of records submitted from sensor modules  CreateUser() – create a user for the system and pick their access level | | Description: | This class is a specialization of the WebUser class. As such, it inherits all the attributes and functions of the WebUser class. | |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | |  |  | | --- | --- | | Class Name: | QualAssurance (specialization of WebUser) | | Attributes: |  | | Functions: | ViewRecentRecords() – view up to date information regarding sensors | | Description: | This class is a specialization of the WebUser class. As such, it inherits all the attributes and functions of the WebUser class. | |

Traceability Matrix

Use Cases for system and system requirements:

Table

Description automatically generated

Table

Description automatically generated

System Architecture and System Design

Architectural Style

The system is developed with component-based architecture in mind. Component-based architecture makes a conscious attempt to split different subsystems and responsibilities into different modules. Within the Quality Data System, there are a number of different subsystems that interact, but are independent of one another. These subsystems include the Sensor Modules, the Database Module, and the Web Module. These three systems record the data, store the data, and display the data respectfully. This creates a separation between points of failure, where a single failure would not likely take out the entire system.

The system is also database-centric, meaning that it was designed with a database in mind for its storage requirements. The application relies on the database to store and deliver the information required for it to function. All of the storage takes place in the database and any actions done take place on data that was delivered from the database. Additionally, having a database simplifies the storage of information and provides a working toolset of data access functions, saving time and resources needed for developing those. Modern databases include management systems that simplify frequent tasks such as indexing and backups. Additionally, modern databases also include ACID properties in their transactions, assuring atomicity, consistency, isolation, and durability in order to protect the information that they store.

Identifying Subsystems

Diagram

Description automatically generated

Mapping Subsystems to Hardware

As previously described above in the subsystem identification, the system utilizes a number of subsystems in order to function. The data acquisition takes place on sensor modules, the data storage takes place on a database module, the delivery of data takes place on a web module, and lastly the user accesses the system utilizing a web browser on a device on the same network.   
  
 The data acquisition subsystem of sensor modules will run on a number of Raspberry Pi devices outfitted with DHT11 temperature and humidity sensor modules. These are connected via the GPIO pin of the devices and utilize a program to gather information from the DHT11 component and deliver it to the database. These Raspberry Pis are more limited in their use, so budget models are more appropriate here.

The data storage takes place within the database module. This is a more powerful Raspberry Pi running a Maria Database instance. The database accepts incoming connections via users that are created within the database. These users are part of a connection function within the web server. The database has indexes for faster sorting, primary/foreign key constraints for data integrity, and has large permanent memory for storing database information for large periods of time.

The web module runs on another Raspberry Pi and is responsible for serving the webpages to the user. This module utilizes an Apache web server to deliver content to incoming connections. It has the all the code for the website stored on it in different files. This is a model of Raspberry Pi similar in power to the database module as it has to run PHP code in order to generate some content for the users.

Lastly, the users connect using their own devices. In a way, they are part of the system because portions of the code from the web module execute within their web browsers. The web browsers display the HTML and CSS served from the web server, but also execute JavaScript functions locally in order to do some manipulation and displaying of data.

Persistent Data Storage

As this is a database backed system, persistent storage is necessary in order to store and deliver data related to user accounts, information about the system, and temperature and humidity data. This information is stored within a database on the database module. The database stores this information locally on a microSD card inserted into the Raspberry Pi hosting the database. A 64 gigabyte microSD functions as the database storage device and allows adequate space for containing the information. Additionally, the database can be directly queried via ODBC connection to download information directly into Excel files.

Network Protocols

The system utilizes network protocols in order to communicate data between systems and to the end user. Multiple subsystems are interacting, and as such are required to use a local-area network in order to communicate with each other. The requirement is that these devices are on the same network. The system expects general, common networking protocols be functional, such as DHCP and DNS for allocating dynamic IP addresses if needed and for finding machines by hostname. The webserver and database server will have static IP addresses for simplicity, but the sensor modules can function with dynamic IP addresses if needed.

The data from the sensor modules is delivered to the SQL database via ODBC or Open Database Connection protocol. The data is retrieved from the database to the website utilizing a connection string as well.

Lastly, the end user connects to the webpage using hypertext transfer protocol, HTTP. Currently, the secure version is not in use, but it is possible that it might be added later.

Global Control Flow

The application can be utilized in many different orders from the user perspective. The only requirement is that the user authenticate into the system. At that point, the user is presented with multiple functions or pages that they may visit. Visiting these pages can be done in any order or can be skipped entirely.

The system is fairly asynchronous. Users can dynamically create data requests by accessing pages that call backend code that queries data out of the database system or by interacting with the elements delivered to their devices. For this portion, there are no time dependencies. For the sensor modules, they are programmed to send a new data record to the database every 5 minutes. This is done via the python script on the sensor itself.

For hardware resources, the response will be divided into two parts: system and user. For system resources, a minimum of one Raspberry Pi is required. The database, web server, and DHT11 sensor component have to reside on a device. The recommended is for these three components to be spread across three separate pieces of hardware, however. Additionally, more than one sensor module can exist to report temperatures in different areas. For the database module, a Raspberry Pi 4 is recommended with a minimum of 1 gig of RAM. It will require a microSD card for local storage of the OS and the database data. Recommended is 32 gigabytes or more to ensure that the database doesn’t quickly grow to being too large in side. Additionally, the system will require some sort of network connectivity either through a wireless network adapter or a physical one. The web server module will require another Raspberry Pi with the same recommendations as above, but the microSD card can be smaller if needed do to it not storing database records on it. Lastly, the sensor modules can be a less powerful version of the Raspberry Pi and require that a DHT11 sensor module be installed via GPIO pins. One consideration to be aware of, however, is that GPIO pins are referenced by an identifier at the system level and these GPIO pins have changed location between Raspberry Pi models. For this system, we utilize pin 4 for the delivery of data from the DHT11 module, but this pin might instead be a voltage carrying pin in another model of Pi. As such, the pin out of different models of Raspberry Pis must be taken into consideration, and if the pinout has changed the code must be changed to reflect that.

Algorithms and Data Structures

Description of Data Structures and Algorithms

The system does not currently utilize any complex data structures or algorithms that require in-depth clarification.

User Interface Design and Implementation

Recent Design Changes

Overall, the initial diagrams that were developed as part of the user interface and design have remained somewhat consistent. A large amount of appearance changes have taken place in regards to style and readability, but its still largely conforms to the diagrams developed as part of the initial interface design. One main difference is, however, is the design and implementation of a collapsible navigation bar system to assist users in traversing the multiple pages. This was done in order to not detract from the information being presented to the end-users while still creating an experience that is consistent and simple from person to person.

Design of Tests

Testing Overview and Methodology

For the system, the design of tests will focus around two subcategories: the technical functionality of the system and the intuitiveness of the system. The technical functionality focuses on testing the implemented use cases and seeing if the system can be broken in anyway. The usability testing focuses on whether or not a user can intuitively utilize the system to accomplish their tasks simply. A failure in the functional testing indicates an issue stemming from programming issues that require addressed. A failure during the usability testing indicates the design is not sufficiently simple and intuitive to support ease of use for new users and may require new design work be done.

A number of use cases were identified in the prior sections. These include viewing single sensor data, viewing sensor dashboard, exporting data, viewing trend data, logging in, changing your password, creating a new user, resetting another user’s password, viewing temperature modules, and creating comments. These use cases all represent things a user would do within the system and will be tested through attempting to enter values or process flows that create in an error with the system, either at the business data level or via generating a programming error. For each case, minimum and maximum values will attempt to be defined, constraints will be tested, and different combinations of workflows will be approached in an attempt to cause an unexpected result. Separately, due to the number of different interacting systems, the temperature sensor Pis also require a fair amount of testing through by allowing them to run for a sufficient period of time to validate the information they are reporting is accurate. They will also be independently validated by having the temperature and humidity levels independently checked.

For the system design and usability, usability studies will take place. A handful of test users will be granted access to the system and in a controlled environment will be given tasks to accomplish with no direction. Their times will be recorded for individual tasks and they will be asked to complete a survey at the end. Through investigation of the individual times and surveys, areas of struggle will be identified and the feedback will be incorporated into the system design in an attempt to create a more user-friendly system that emphasizes simplicity of use.

Project Management and Plan of Work

Project Coordination and Project Report

Currently, a number of use cases have already been fully implemented into the system. Viewing single sensor data views, exporting data, logging in, changing your password, creating a new user, resetting another user’s password, and creating comments have all been fully implemented into the system. What currently remains is the fine tuning of the systems for viewing trend data, the sensor dashboard, and viewing temperature modules. Additionally, another main goal is the implementation of the design choices created for the system. Originally, the functionality was tested prior to the implementation of the final design choices as part of a ‘back-to-front’ methodology. As the use cases become fully implemented, more time is directed towards ensuring that the system is delivering a visually appealing interaction that is simple for the users to utilize.

Plan of Work

A picture containing diagram

Description automatically generated

Above is the remaining plan of work for the project. The work is broken into three remaining tasks: design, testing and UI, and Cleaning and Stretch Goals. The design category includes the finishing of the implementation of the new design schemes to make the site more appealing to end users and simplifying navigation. The testing and UI portion refers to the testing portion of the application and the implementation of the feedback received from it. The last portion allows time for polish to be applied to the system and additional goals be implemented if time allows.

Works Cited

Tilley, S. R., & Rosenblatt, H. J. (2017). *Systems analysis and design*. Course Technology Cengage Learning.