

## ***Project juni 2017***

### ***02324 Videregående programmering***

Project name: CDIO Final

Group number: 14

Due date: Friday, 16/6 2017 at 12:00

Conceive Develop Implement Operate project (CDIO) Final:

### **Automatic H.A.C.C.P Program**

#### **Group 14**

Rasmus Blichfeldt	s165205
Casper Bodskov	s165211
Lasse Dyrsted	s165240
Michael Klan	s144865
Mathias Larsen	s137055
Timothy Rasmussen	s144146



Danmarks Tekniske Universitet DTU

#### **Supervisors:**

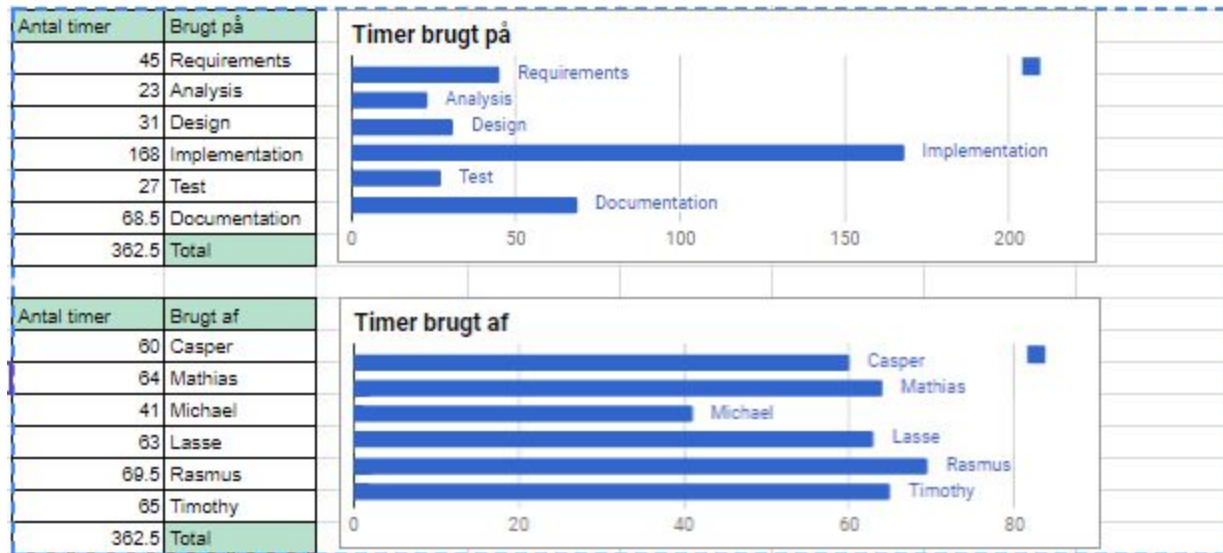
Stig Høgh

Finn Gustafsson

Ronnie Dalsgaard

Bjarne Poulsen

## Time table and tasks



## Link to website

[http://dyrsted.duckdns.org:8888/CDIO\\_WebProject/](http://dyrsted.duckdns.org:8888/CDIO_WebProject/)

Login:

Username for admin = "admin" (without " ")

Password for admin = "1" (without " ")

Username for regional manager = "region"

Password for regional manager = "1"

Username for head chef = "headchef"

Password for head chef = "1"

Username for chef = "chef"

Password for chef = "1"

# 1. How to run the programs

1. Make sure you have internet access
2. Download the folder 14\_Final
3. Run the .bat file called run. This opens the local server and dummy sensor.
4. Open browser
5. Navigate to [http://dyrsted.duckdns.org:8888/CDIO\\_WebProject/](http://dyrsted.duckdns.org:8888/CDIO_WebProject/)
6. Here you have the options to create a new user, (note that a new user needs to be assigned a department before being able to view info after logging in. this is done by a user with a higher user class) and log in. the other pages also work but have placeholder text.
7. Logging in takes you to the dashboard
8. Here info from the database is displayed (info depends on user class)
9. Info from sensors and alarms are auto updated in real time
10. When done exploring logout. This takes you back to the frontpage.

# Abstract

This report serves as the documentation of the work done on the system created to make self-monitoring easier for restaurants and other businesses related to food. The system includes sensors which will transmit data to a local server which will then transmit any processed data to the database “cloud”-solution. The system will also consist of a webpage which will automatically update the sensor values on from the user’s local server. The local server will be located at the restaurant and must therefore only transmit data to the cloud and not access tables from it. The webpage will serve as the gateway for the end user to check status of his/her kitchen as well as see any current workers, sensor values and other relevant information. This project is made to satisfy the assignment and will therefore not represent the fully functional system since it’s a prototype of the functionality related to the subject and only the functions which were deemed doable and necessary for the assignment. In conclusion the program runs smoothly and can be exported as jar files which can run from a bat file making the execution of the program simple.

# Table of contents

<b>1. How to run the programs</b>	<b>2</b>
<b>2. Introduction</b>	<b>6</b>
<b>3. Project Analysis</b>	<b>7</b>
3.1. H.A.C.C.P requirements	7
3.1.1. Cool chain	7
3.1.2. Deliveries	7
3.1.3. Fridges, freezers	7
3.1.4. Reheating	7
3.1.5. Keeping products hot	7
3.1.6. Keeping products at room temperature or semi cooled	8
3.1.7. Cooldown	8
3.1.8. Dishwasher	8
3.1.9. H.A.C.C.P. strategy	8
3.2. Business logic	8
3.2.1. The problem	8
3.2.2. The product	9
3.3. Requirements specification	11
<b>4. System Design</b>	<b>13</b>
4.1. Use Case diagram	13
4.2. Use case descriptions	14
4.3. Domain models	19
4.3.1. Dummy Sensor domain model	19
4.3.2. Java Server domain model	20
4.4. Design class diagrams	21
4.4.1. Dummy Sensor class diagram	22
4.4.2. Java Server class diagram	23
4.5. Schema	24
4.6. Our considerations about the database	25
4.7. Relation Schema diagram	26
4.8. Functional database setup	27
4.9. Sensor Build	28
<b>5. Test</b>	<b>29</b>
5.1. Dummy Sensor test	29
5.2. Physical sensor test	30
5.3. Local server test	31

5.4. Database test	32
5.5. Website test	32
<b>6. Conclusion</b>	<b>33</b>
<b>7. Appendix</b>	<b>34</b>
7.1. Appendix 1	34

## 2. Introduction

This report serves to satisfy our CDIO assignment. The report will include all diagrams we deem necessary to complete the assignment efficiently while showing our thought process. The assignment was tailored to suit our own aspirations for this project. The system will be done according to the assignment which will set it up for further development. Our requirements for this assignment is to create the physical sensor prototype to test, a java application to function as the local server, a database to function as a cloud solution and a website to enable and limit access to the database for the end user. The report will contain diagrams to show the structure of the program, as well as use cases to show the relation between the code and the actor. We will also use various diagrams to show the database structure to show the structure despite not being the main subject for this assignment. It should satisfy the requirements set in our requirements specification and will follow the guidelines set in our last semester as well as include the new material introduced in this semester like our use of javascript, HTML and CSS for the website as well as some PHP and arduino programming despite being outside the syllabus of the subject.

## 3. Project Analysis

### 3.1. H.A.C.C.P requirements

H.A.C.C.P (Hazard Analysis and Critical Control Points) is the international standard for food safety, describing all aspects of food safety from farm to consumer including all vectors to the consumer. Our product focuses on self regulation in restaurants, canteens and convenience stores selling perishable products to the end consumer. To meet the requirements for self regulation, the following areas are critical.

#### 3.1.1. Cool chain

The cool chain entails that all products must be kept at an appropriate temperature from farmer to end consumer. For example meats needs for be kept between 0.1c° and 5c°.

#### 3.1.2. Deliveries

When receiving perishable products it is required that a temperature reading of one or more products are documented, and is within the appropriate values. This is done to ensure that the cool chain have been maintained.

#### 3.1.3. Fridges, freezers

The temperature of all units must be documented 1-2 times daily and be within the appropriate ranges described in appendix 1. If the temperature have been outside of that range for a prolonged period of time, for example if the appliance have been broken, unplugged or similar situations, it must be documented what have been done to insure the food safety.

**Fridges** must have a temperature range of 0.5 - 5c°

**Freezers** must have a temperature lower than -18c°

#### 3.1.4. Reheating

Any food processing facility that reheats products must document that the process have taken less than 3 hours and the concluding temperature must be over 75c°.

#### 3.1.5. Keeping products hot

If a food processing facility produces food that is kept hot ex. in a buffet the temperature of the products must be over 65c° and must be used or discarded after 3 hours. This must be documented by sampling once daily.



### 3.1.6. Keeping products at room temperature or semi cooled

Most products can be held at room temperature or be cooled down to a temperature of over 5c° for up to 3 hours before decaying if the food processing facility sells to the consumer market. For example this rule is used in canteens. This is sometimes documented depending on the location's function, a canteen is often open for less than 3 hours and is therefore exempt from documenting this aspect.

### 3.1.7. Cooldown

Any product must be cooled from 65c° to under 10c° in under 3 hours. This must be documented by sampling once daily.

### 3.1.8. Dishwasher

All dishwashers must run at a temperature of over 85c° this must be documented once daily.

### 3.1.9. H.A.C.C.P. strategy

All food processing facility must be Registered with "Fødevarestyrelsen" or the equivalent government agency in the respective countries. As part of this process the facility must write a H.A.C.C.P strategy describing how to handle food safety according to the rules stated above.

## 3.2. Business logic

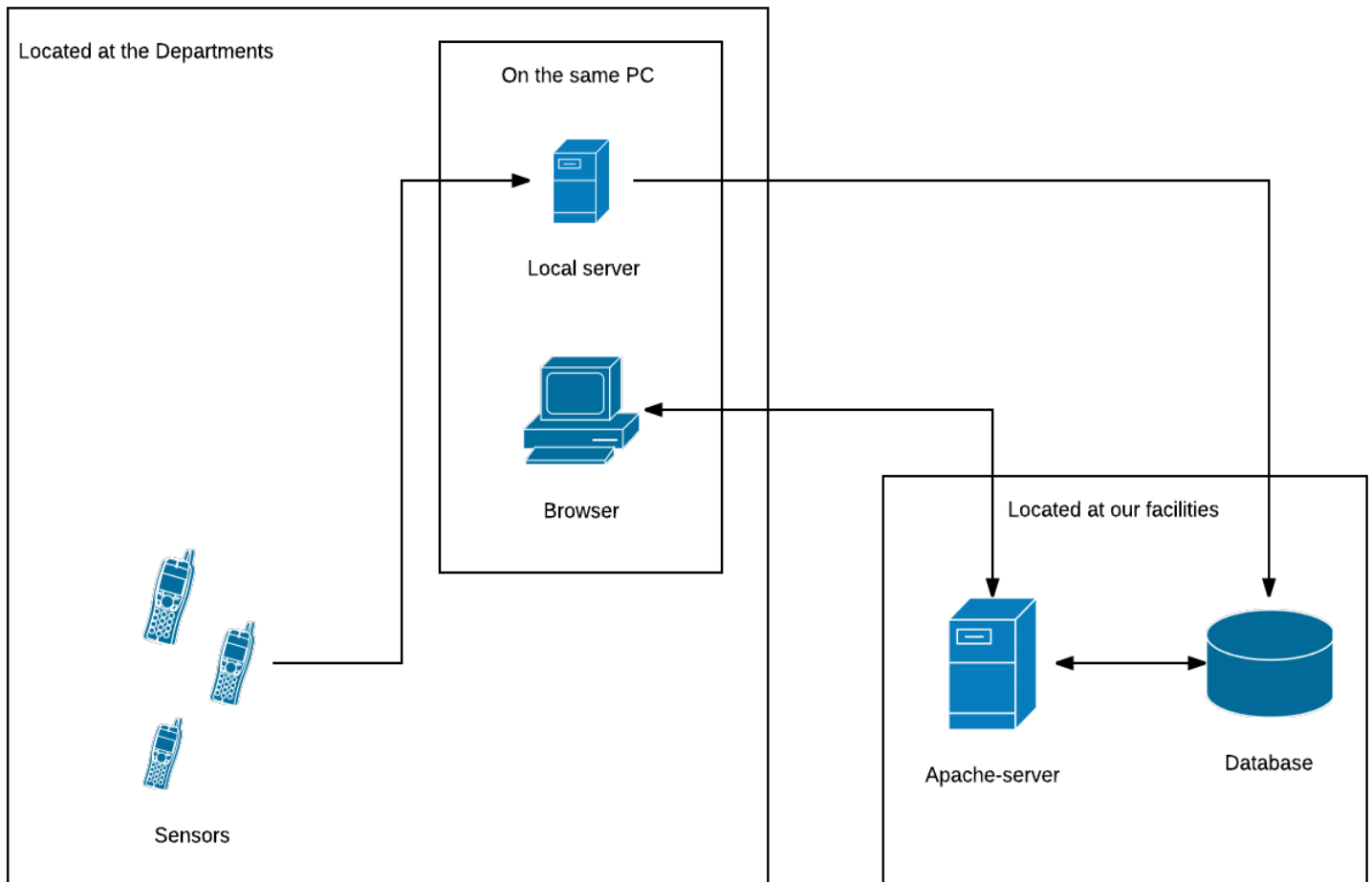
We want to start a company called "Food CDIO" (placeholder name). Food CDIO delivers software solutions to the food industry. Our first product focuses on monitoring the food and warning users when jeopardizing the food edibility.

### 3.2.1. The problem

Many restaurants does not document the self-monitoring to the specifications that the food Administration "Fødevarestyrelsen" requires. Often chefs forget to make the daily documentation sometimes for weeks. Therefore the data is useless from the perspective of food Administration. If the documentation is not filled out when food Administration makes an unannounced controle visit the lack thereof can result in a low food safety rating. Due to the fact that consumers takes this rating into account when choosing where to eat the problem can cause significant financial losses to the restaurant.

### 3.2.2. The product

Food CDIO will use general and specialised temperature sensors that communicate over wifi to document most of the temperatures automatically. The data will be sent to a local server in which the server analyses the data and send to our database. As part of the analysis alarms will be triggered if temperatures are out of the appropriate ranges. This can save the facility a substantial amount of money as failing appliances and forgetfulness (ie. if someone forgets to close a fridge probably) will be discovered before products have to be discarded.



This diagram shows the connections between each component and their locations.

**The local server** program will run on a computer at the facility that have bought our product. The purpose of the local server is to receive data from the sensors and send it to our database located at our own facilities. It is also responsible for triggering all alarms except those that relate to lost connection between the local and our servers. If the facility does not have a computer we will

provide a specialised computer ex. a raspberry pi at a competitive price. However we expect that most facilities have a computer already.

### **The sensors:**

Will be built and programmed to monitor and document the different aspects of the kitchens daily activities and if they meet the requirements set as described in the H.A.C.C.P. requirements section above. Most of the documentation will be done automatically and the remaining documentation will be semiautomatic with reminders being sent to staff by phone, email, website or light/sound depending on the customer's needs. We will strive to make everything automatic.

#### **Sensor Operating Cost:**

Since the sensor runs on a battery charge it is a problem to change the battery. We cannot replace the power supply with a charger or directly to a wall outlet since it would compromise the idea that it's small and won't obstruct the work of the chefs. Therefore it is necessary to consider the lifetime of the battery as well as how to prolong said lifetime. That is why we've implemented the sleep function which enables the sensors to conserve power and therefore prolong the lifetime. This disables some of the functions of the sensor and it will only wake when needed.

**The web server** does not only serve as a way to access the website it also acts as a wall between the users and our database. This increases the security and integrity of our database. The web server will also be responsible for triggering certain kinds of alarms for example if a facility have not sent any data for a certain amount of time, an alarm will be set off prompting the user to reestablish the connection.

**The website** will be used to show all the collected data in a visually pleasing way that makes it easy to evaluate the situation without any instructions. It will also show the H.A.C.C.P. strategy, alarms triggered in the given kitchens, news related to changes in legislation relating to the food industry, callback orders, special dates and similar information. If a company owns several facilities a regional manager can view all data for all facilities.

**The website flow** is how the user accesses our website, where the user will first encounter our front page. The user will be able to access different tabs (frontpage, product page, about us page, contact page, create user, login page). The user can choose to login or create a new user profile when first entering the website. When creating a new user profile, the website will require that they have a valid email, unique username and password, otherwise the creation will not succeed. When attempting to log in with an existing user profile the website sends a request to the database which then checks if the user exists in the database. If the user has entered a valid username and password, he/she will be granted access to the dashboard.

If the password is wrong you get the same error message as if you wrote the username wrong. An alternative solution would be to make an error that showed the user that his password is typed wrong but his username is correct. However this makes bruteforcing a user a lot easier since you

don't need two fields of information after finding the username which is why it's more secure to have a single error showing the credentials are incorrect.

When the user gets access to the dashboard, the website will automatically show his/her kitchen data from the database in the form of graphs and values. If the user is connected to more than one kitchen then there will be a menu to select which kitchen data should be displayed on the dashboard. Other than the different values and graphs there will be a tab to print a formal report showing the required values for the "Fødevarerstyrelsen" which the user can print and hand over when inquired to do so.

#### **Security:**

Since there's personal data stored in the database we want the cloud data to be inaccessible from the local server. The only function it can have is the ability to transmit data. This hinders people without authorization from altering data in the database from the local server.

Currently little effort have been directed at making the web site secure as it is just a prototype. When communication with the database procedures, views and functions are used to limit the acces. This also serves to hide the structure of the database.

### **3.3. Requirements specification**

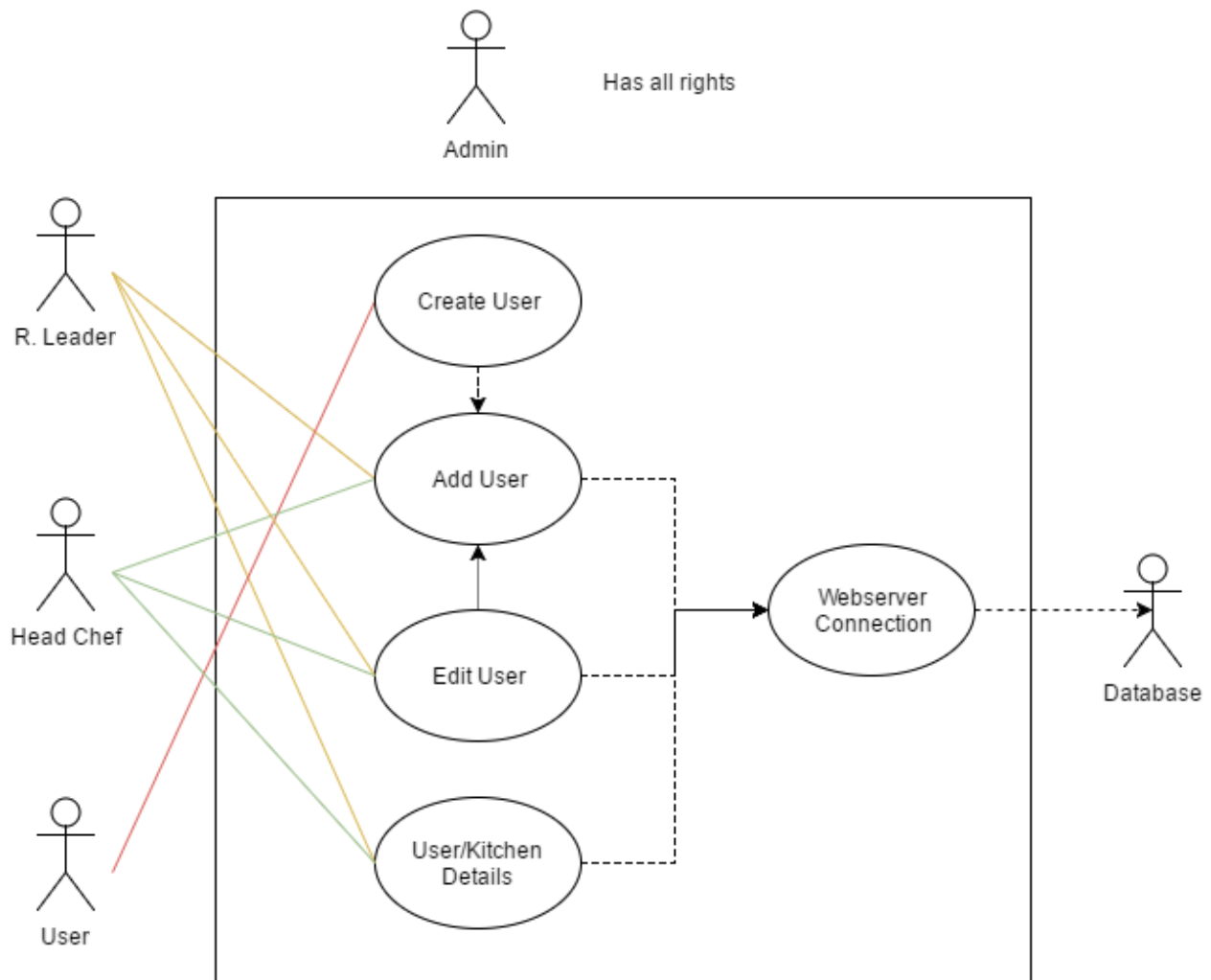
From the project analysis we have created these requirements specification for out system.

1. A database shall be created
2. An arduino temperature sensor shall be used as a prototype for the final sensor
  - 2.1. It shall have an ID
  - 2.2. It shall be able to measure temperatures
  - 2.3. It shall be able to transmit data across wifi
3. Five types of dummy sensors shall be created
  - 3.1. Each sensor shall generate random numbers
  - 3.2. Each sensor shall able to test alarm functionality
    - 3.2.1. One of which shall be for refrigerators
    - 3.2.2. One of which shall be for keeping hot
    - 3.2.3. One of which shall be for reheating
      - 3.2.3.1. Shall have a method to increase temperature over time.
    - 3.2.4. One of which shall be for cooling down
      - 3.2.4.1. Shall have a method to decrease temperature over time.
    - 3.2.5. One of which shall be for freezers
4. A local server shall be implemented
  - 4.1. It shall be able to receive data from sensors assigned to it
  - 4.2. It shall be able to transmit data to our database
  - 4.3. It shall be able to trigger alarms assigned to it
5. A website shall be created

- 5.1. It shall include a login feature
- 5.2. It shall include the following user types
  - 5.2.1. Chef
    - 5.2.1.1. Shall be allowed to view alarms in his/her department
    - 5.2.1.2. Shall be allowed to view data from each sensor in his/her department
    - 5.2.1.3. Shall be allowed to view users in his/her department
    - 5.2.1.4. Shall be allowed to view information (including all information from the address table in the database) about his/her department
  - 5.2.2. Head chef
    - 5.2.2.1. Shall be allowed to do all of the above mentioned actions
    - 5.2.2.2. Shall be allowed to manage(delete, update, add) users in his/her department
  - 5.2.3. Regional manager
    - 5.2.3.1. Shall be allowed to do all of the above mentioned actions
    - 5.2.3.2. Shall be allowed to view all kitchens in his domain
  - 5.2.4. Admin
    - 5.2.4.1. Shall be allowed full access to any and all functions of the website
- 6. Tests
  - 6.1. The local server shall be tested
  - 6.2. The sensor shall be tested
  - 6.3. The dummy sensor shall be tested
- 7. Non-functional requirements
  - 7.1. The Local Server shall be programmed in java, and be runnable on Windows.
  - 7.2. The Arduino sensor shall be programmed in C.
  - 7.3. The Database shall implemented with MySQL.
  - 7.4. The Website shall be programmed in HTML5, CSS, Javascript.
  - 7.5. The Server side language shall be PHP.
  - 7.6. The program shall run on the computers available in the data bars on DTU campus Lyngby.
  - 7.7. Everything in the report as well as the code shall be written in English (UK).
  - 7.8. The program shall be written in UTF-8.
  - 7.9. All work on the code of the program shall be done in coherence with Github and commits shall be done to the designated repository:
    - 7.9.1. Once every hour
    - 7.9.2. Every time a smaller objective or piece of code is finished.

## 4. System Design

### 4.1. Use Case diagram



The use case diagram shows how interactive the program is and how the user affects the program. In other words the actor's interaction with the program. This diagram has multiple actors since there's multiple "roles" which enable specific functions. To avoid confusion the actual users of the program is set as different actors which serves to show what kind of user got access to which functions. The use case descriptions will explain more in-depth as to what happens during each of the interactions whereas the diagram is more simple and only serves to give an overview of which actions are available to whom. The use case diagram is a static model since it has no runtime depiction and only shows how the user interacts with the program. The use case descriptions however has a flow and could be described as dynamic despite not being models.

The admin has access to everything just in case something goes wrong or needs to be manually altered. However this is not any desired “flow” of the system, since it’s only meant for testing and emergencies, hence no use case descriptions are made to show these connections. This is shown by dotted lines in the diagram.

## 4.2. Use case descriptions

Name	Web server Connection
Identifier	UC1
Description	How the system communicates with the database.
Primary actors	Program
Secondary actors	None
Preconditions	The Local Server’s timer says it’s been 5 minutes since last value was submitted.
Main flow	<ol style="list-style-type: none"> <li>1. The program establishes a TCP connection the the sensor.</li> <li>2. The program requests the value from the connected sensor.</li> <li>3. The program receives the value from the connected sensor.</li> <li>4. The program closes the connection with the sensor.</li> <li>5. The program establishes a connection to the database.</li> <li>6. The program sends the data to the database.</li> <li>7. The program closes the connection with the database.</li> </ol>
Postconditions	The program resets the timer to count 5 more minutes.
Alternative flow	<ol style="list-style-type: none"> <li>1. If the program can’t establish a connection to the sensor it throws an exception and shows an error.</li> <li>2. If the program can’t establish a connection to the sensor it throws an exception and shows an error.</li> </ol>

Name	Add User
Identifier	UC2
Description	How to designate a user to a kitchen
Primary actors	Regional Leader, Head Chef
Secondary actors	None
Preconditions	There is at least one (1) head chef or regional manager and one (1) user
Main flow	<ol style="list-style-type: none"> <li>1. Login as a regional manager / head chef</li> <li>2. Search for the username of the desired user</li> <li>3. Press assign button (will assign the user to the same kitchen as the regional manager / head chef)</li> </ol>
Postconditions	Is removed from the list of unassigned users.
Alternative flow	If no users are in the database when “add user” is selected, the system informs the actor and does nothing else.



Name	Edit user
Identifier	UC3
Description	How to Edit a user
Primary actors	Head Chef, Regional Leader
Secondary actors	None
Preconditions	There is at least one (1) user in the database.
Main flow	<ol style="list-style-type: none"> <li>1. Mark the edit radio button on the desired user</li> <li>2. Select "edit"</li> <li>3. Change the desired table(s)</li> <li>4. Select "submit"</li> </ol>
Postconditions	The actor is sent back to the dashboard.
Alternative flow	If no users are in the database when "edit" is selected, the system informs the actor and does nothing else.

Name	User/Department Details
Identifier	UC4
Description	How to view user and department info
Primary actors	Admin, Regional leader, Head Chef, Chef
Secondary actors	None
Preconditions	Frontpage of the website is open
Main flow	<ol style="list-style-type: none"> <li>1. Actor logs in</li> <li>2. Browser opens the dashboard and shows info of users and the main the main department the actor belongs to</li> <li>3. If the actor belongs to several departments. <ol style="list-style-type: none"> <li>3.1. Actor clicks the select an option bar at the top of the page and selects a department</li> <li>3.2. The browser shows info of all selected departments</li> </ol> </li> </ol>
Postconditions	none
Alternative flow	none

Name	Create User
Identifier	UC5
Description	How to create a user
Primary actors	User
Secondary actors	None
Preconditions	Frontpage of the website is open
Main flow	<ol style="list-style-type: none"> <li>1. Actor clicks on create user in the top menu</li> <li>2. Browser opens the create user page</li> <li>3. Actor enters his personal details then clicks the submit button</li> <li>4. Browser calls the create user procedure</li> <li>5. Database creates tulip in user table</li> </ol>
Postconditions	The actor is sent back to the frontpage
Alternative flow	

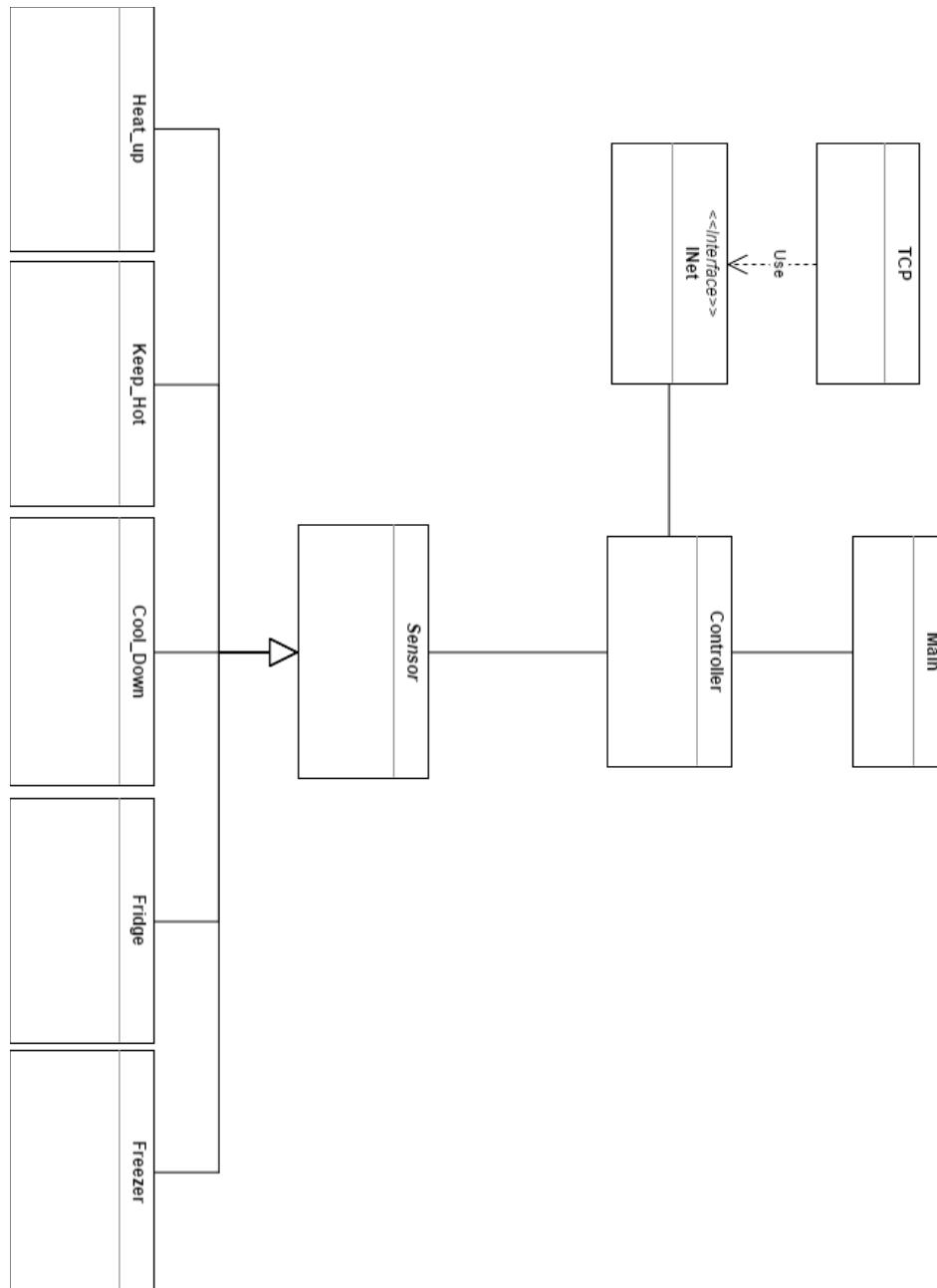
The use case description takes specific scenario and shows the flow of the program when the “actor”, in this case the program, interacts with different parts of the system. The description shows the steps of the flow as well as any actors, pre- and postconditions in one sleek table. Our use cases does not describe the process of all the admin’s abilities since he/she can do anything. The admin needs to have right to edit or add or delete anything from the database so it’s possible to test and check all functions as well as manually remove customers who might forget to delete their own accounts. However these are not in the descriptions since it isn’t the main flows of the program. The admin should not have to use any of the functions outside of those described if everything runs smoothly and everyone uses the program as intended. However that is usually not the case and therefore the admin has all rights as a precaution rather than a flow. The functions would work similarly even if it the admin had to do something instead of the designated actor.

## 4.3. Domain models

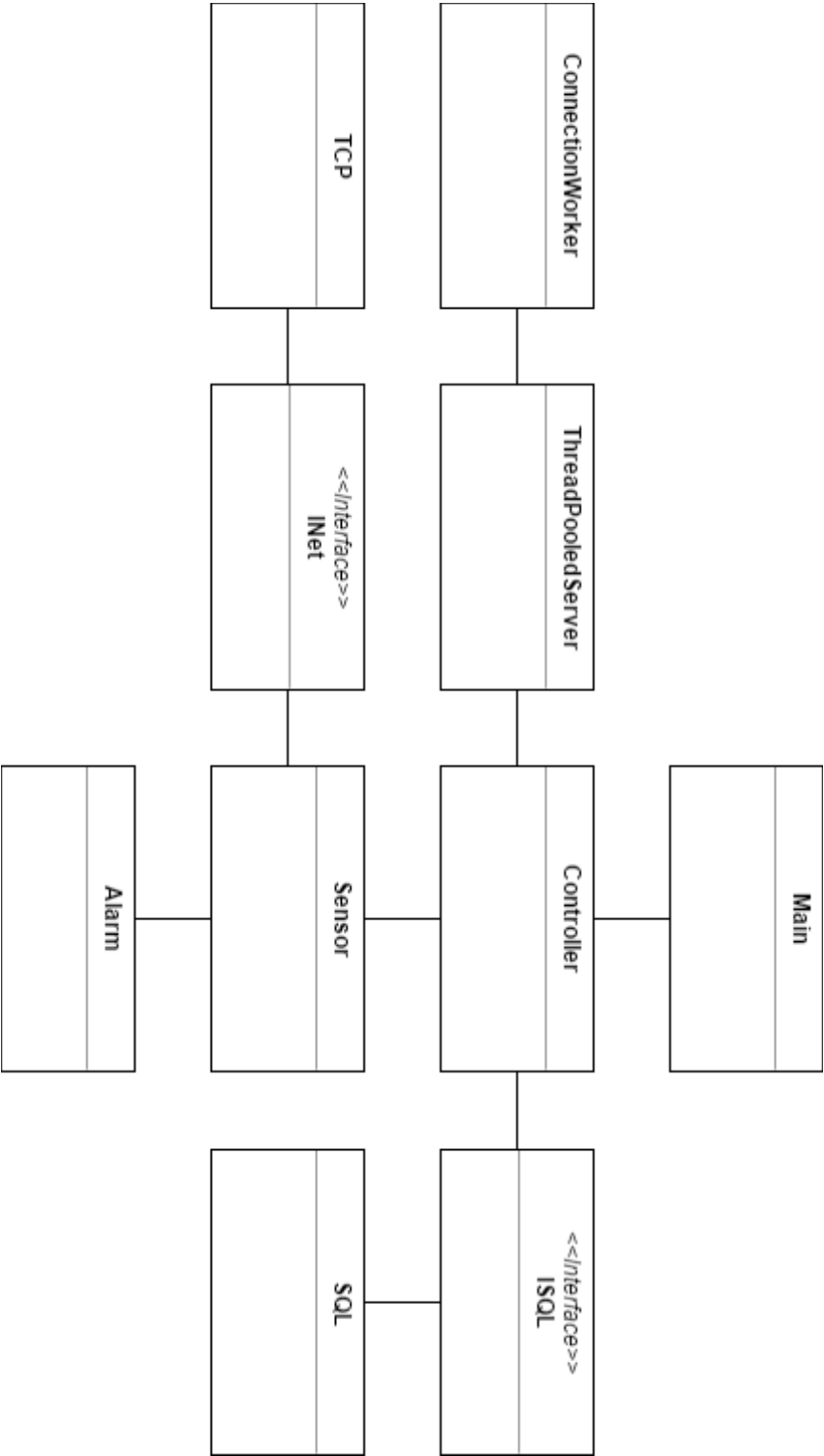
The domain model is a static model which depicts the structure of the code in the simplest manner so the code can be explained to eg. business men with little to no understanding of the actual code. We use this as a precursor to the class diagrams since we have no one we need to explain the structure to, so making it a tool for ourselves were a lot more useful.

This domain model shows the local server structure and with how the classes are connected.

### 4.3.1. Dummy Sensor domain model



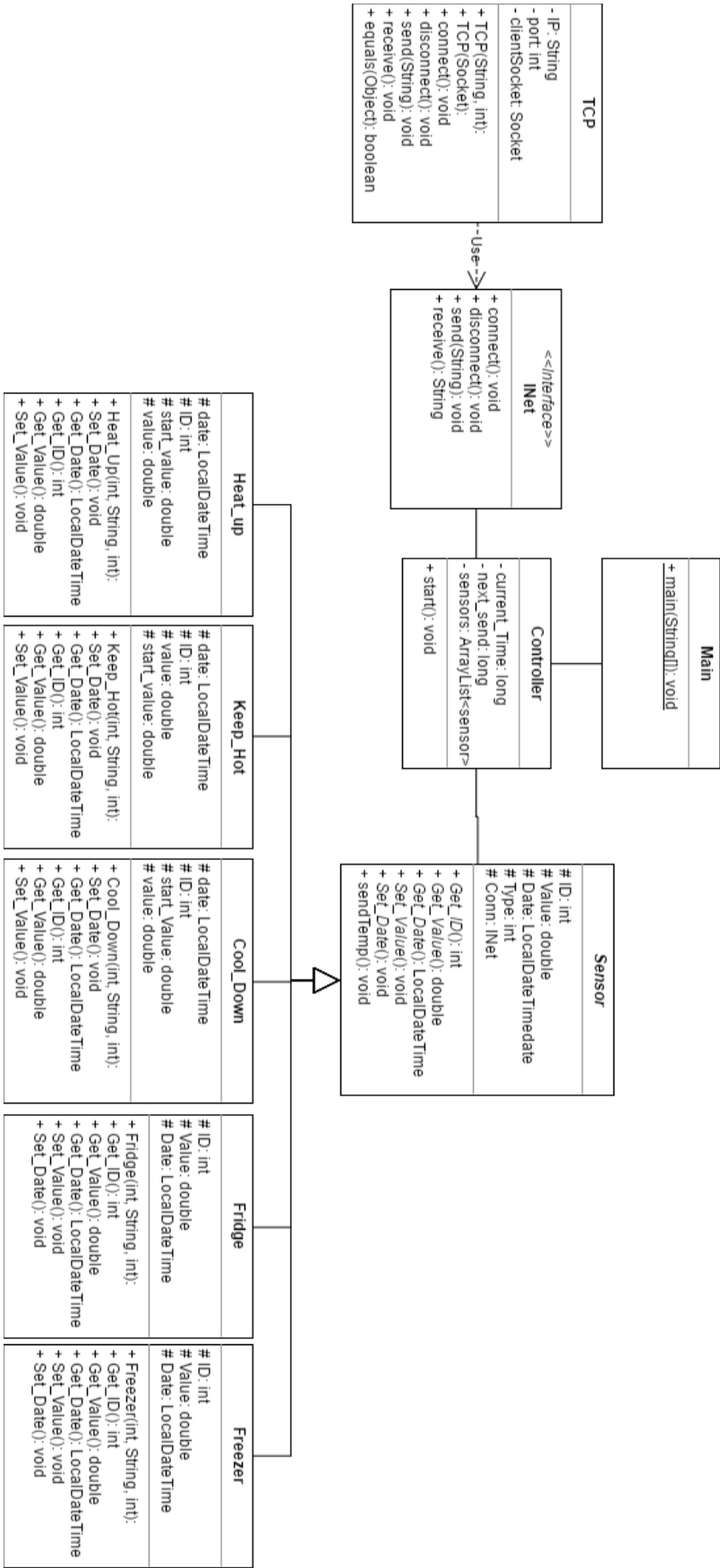
4.3.2. Java Server domain model



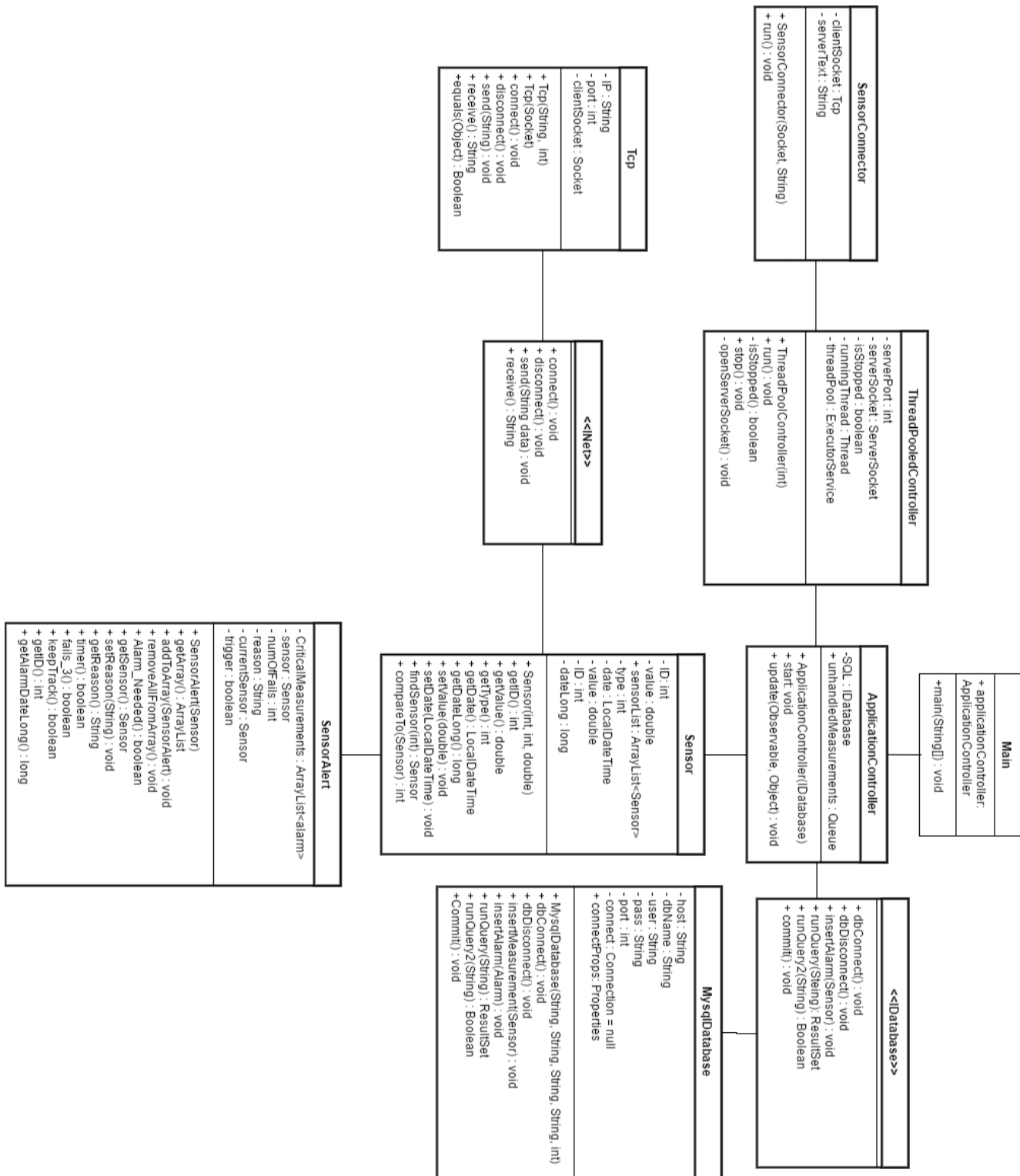
## 4.4. Design class diagrams

The design class diagram is an exact representation of the program. It is static like the domain model but far more detailed and contains all the information necessary to code the program from scratch. It is used to explain or show the program code more visually than looking in the source code without losing information about the specifics of the code. We use momentary analytical class diagrams to produce the final class diagram shown in the report. The analytical diagrams are solely used as a development tool. They are adjusted whenever the code is changed and will eventually form the design diagram. It makes our work flow a lot less prone to errors and waste of time since we can detect errors by looking at the diagram before starting the coding process of each individual class. The design diagram is then the final draft which we then try and make as coherent as possible while keeping all the class information.

4.4.1. Dummy Sensor class diagram



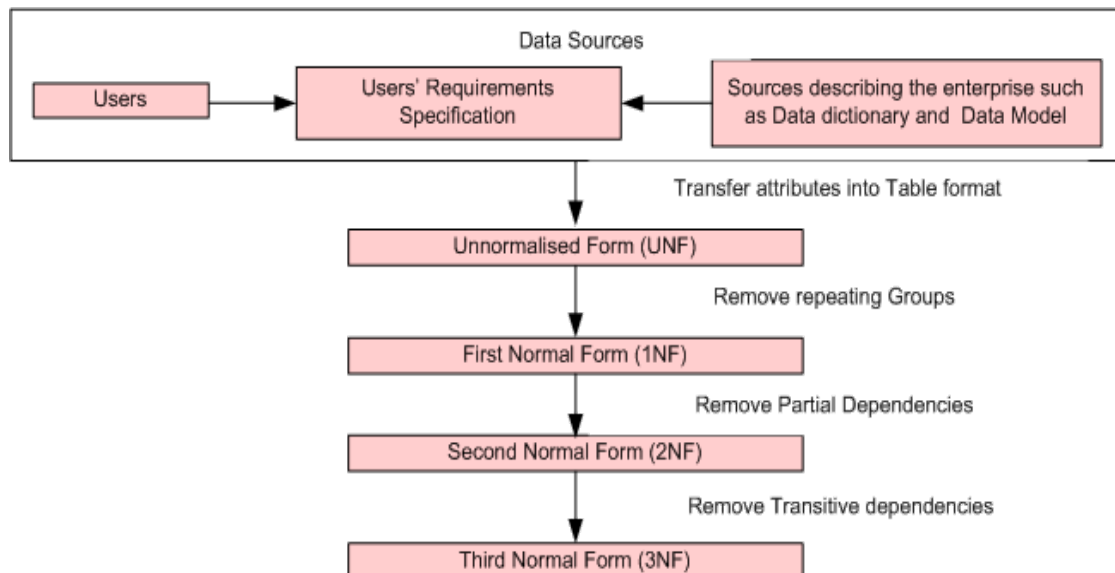
4.4.2. Java Server class diagram





## 4.5. Schema

For the design of our database we want to make a schema diagram. However we made the diagram with an unconventional approach. Our group is very experienced in class diagrams for program codes and making these requires the classes to be transferred into table format which we do without a second thought since we've done it so many times. This posed a problem later since the correct approach to make a schema requires you to start with "data sources" in other words, pure text versions of the tables and from those create the actual tables which will become the final schema diagram after enough of the normal forms have been satisfied. Our method simply made the tables as we thought they would be with all the primary and foreign keys we could imagine we needed. This worked out only because the diagram was rather simple in terms of normalization but resulted in problems later in the process.



The correct approach to making a schema diagram showing each of the normal forms and what is required for the normal form to be satisfied.

Our schema has the following tables:

Sensor(Sensor\_ID, Type, Department\_ID, Appliance\_ID)

Measurements(Sensor\_ID, DateTime, Value)

Alarm(Alarm\_ID, DateTime, Sensor\_ID, Reason)

Department(Department\_ID, Address\_ID, Name, Type, Owner\_Department\_ID)

Address(Address\_ID, Street, House\_Num, Country, Zip, State)

Employed(User\_Name, Department\_ID)

User(User\_Name, Name, Email, Phone, Role, Password)

## 4.6. Our considerations about the database

In the process of making the schema diagram satisfy the third normal form we met some problems due to our creative process described in the “Schema” section. First off we found out that we had misinterpreted the advice from our instructor that each table had to have a “relation table” in between. This meant we had many unnecessary tables. For example we had a table between Sensor and Alarm called Triggers and one between User and Log\_In called Credentials. The Log\_In was also removed since the User\_Name had to be unique and the password didn’t so using the User\_Name as an ID for the User table meant we could merge the two leaving the password as an attribute like the User’s Name and Phone.

Once we realised this mistake we proceeded to create what would become the first real schema diagram. This diagram had a Kitchen and Office where there’s a Department now since making the two separate tables meant the User had to input NULL values and had transitive dependencies which results in it not following the third normal form. The idea to make a collective Department would make it less redundant since the Kitchen “Name” and Office “Name” could be moved into a single Name. To get the name of the Kitchen and Office individually you would have to use a View to see the correct Office or Kitchen name if the User is connected to both. We would also need an Owner\_Department\_ID to self reference the Department\_ID meaning if the Viewed department is a kitchen is could show what office that owns said kitchen. In case of an office the office would own it self. That’s why the name “Department” was chosen, since it’s a really broad way of saying office and kitchen.

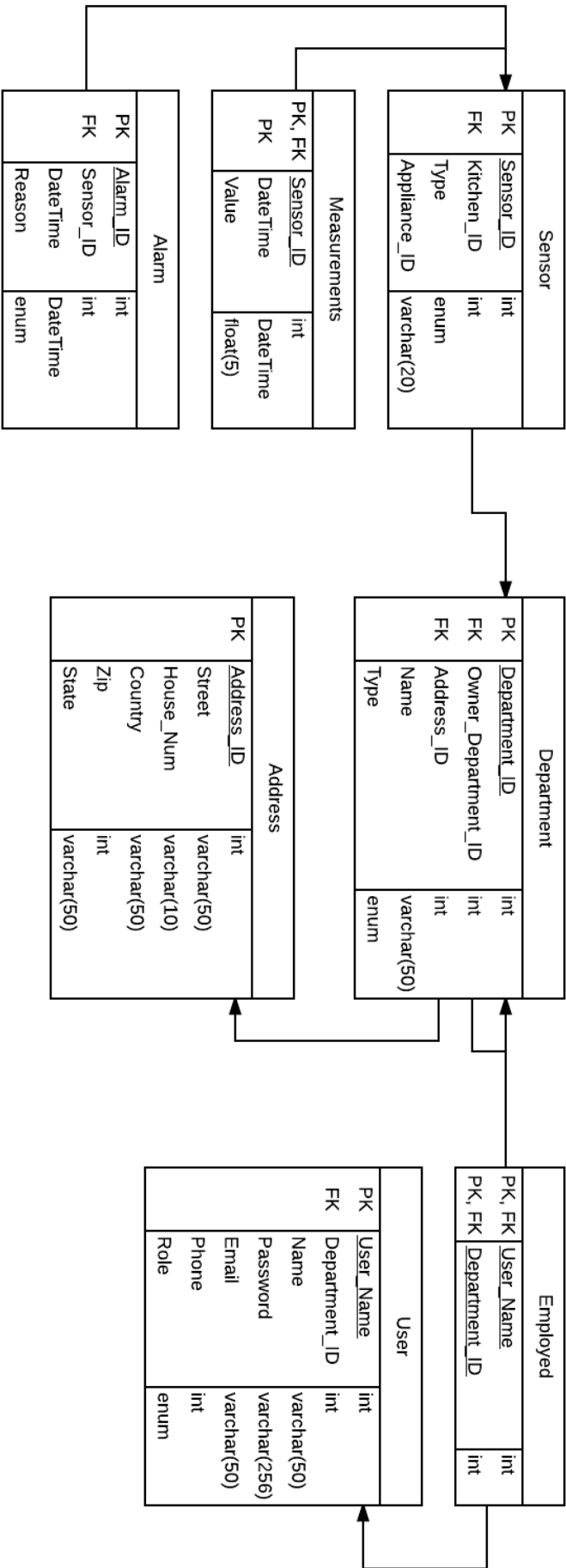
## 4.7. Relation Schema diagram

To make a schema diagram from an ER diagram you need to:

1. Convert complex attributes to simple attributes
2. Convert non-binary relationships to binary
3. Convert entity sets to relation schemas and...
4. Convert relationship sets to relation schemas

After we did that we attempted to create a schema diagram which would serve to graphically describe the database structure in a more precise and detailed manner than the ER diagram. However we did not succeed and went back and forth between our ER and schema diagrams to come up with our final diagrams.

The schema diagram shows our database structure and it used to create the database or show it without showing the actual database and can be compared to the design class diagrams in UML since it's targeted at the developers and not the buyers of the final program.



## 4.8. Functional database setup

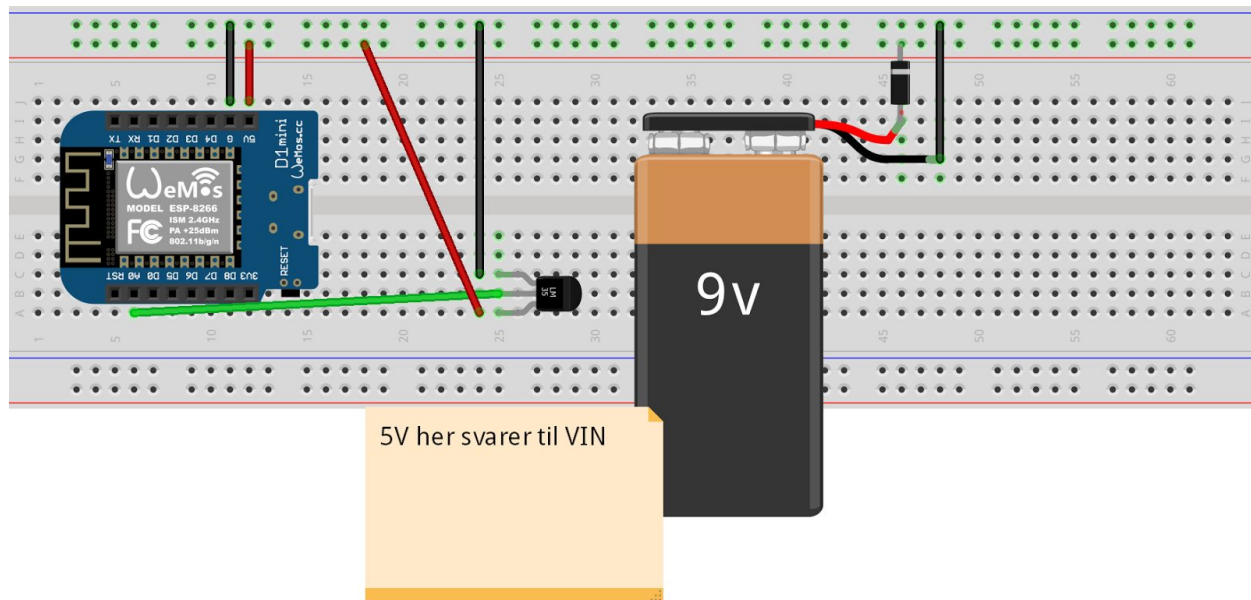
The database is located on a private server at one of the group members home. The database is a MYSQL type. It consists of 7 tables as seen in the schema diagram. There's also 3 views to check alarm values, make lists of all the departments and their assigned users as well as the amount of sensors in a department. The procedures are best represented by the use case descriptions. They keep most the functionality of the database since all edits to personal data are through procedures stored in the database. We used a database managements program to create all the tables in the database which made the implementation easier.

The database functions as the raw data holder where the java server does most of the handling of data. The database procedures and functions we've established are usually called from an interface like the website or directly by the admin if necessary.

## 4.9. Sensor Build

For the sensor we used a modified arduino ESP8266 with the ability to access a wifi network and transmit data to it. It also has the capability to receive and calculate the analog data from the sensor. The temperature sensor we are using in the prototype is a LM35 standard temperature sensor. It's only meant for the refrigerators since it can only handle temperatures between 2 and 150 degrees celsius. Therefore a different temperature sensor is needed for further development. It works by sending an output voltage to the arduino through the analog pin and the arduino will then calculate the temperature reading from the analog value. The measurement can be slightly off linear to the ambient heat around the temperature sensor but it's miniscule ( $\frac{3}{4}^{\circ}\text{C}$  at  $2^{\circ}\text{C}$  or  $150^{\circ}\text{C}$ ) and therefore not significant enough to compromise the functionality in terms of the required precision. We also use a 9V battery as a power supply and a small diode to make it a bit more failsafe if the 9V were to be inverted accidentally.

Below is a representation of the setup.



fritzing

Since our arduino is modified it doesn't appear on regular part lists and therefore we had to use a regular ESP8266. Note that the only relevant change is the size and that we use a VIN instead of the 5V input. A rough estimate of the lifespan with a normal 9V battery, is about 12 hours.

## 5. Test

### 5.1. Dummy Sensor test

We made a dummy sensor that simulates the our real sensor. This is In order to test if our system can handle multiple measurements at once, and to test the different kinds of sensors we would have to have.

Since we only have 1 physical sensor we had to make dummy sensors so that we could test the different types of sensors we would have, instead of reprogramming our sensor for every type we would like to test. This has the added benefit that we can create as many as we want of different types and at the same time, in order to better test our system.

The different types of dummy sensors, which reflect the types of physical sensors we would have, are as follows:

- Fridge
- Freezer
- Heat\_up
- Cool\_down
- Keep\_hot

We can divide these into 2 subgroups, those that involve time and those that do not.

Fridge and Freezer both require to remain within a constant temperature interval).

They will only trigger an alarm once the temperature falls above or below their restrictions.

Heat\_up and Cool\_down are both required to meet a certain temperature within a given time frame. They will trigger an alarm if the time runs out and they haven't reached the given temperature. They will end prematurely when they reach the designated temperature before the time is up.

Keep\_hot requires the sensor to stay above a certain temperature for a certain time.

It will trigger an alarm if either the time runs out or the temperature drops below the given limit.

As the dummy sensor was made to test the system, no junit test of the dummy sensor itself was made. The generation of semi random numbers (within a specified range) was tested and all those numbers were found to give the desired results. The connection to the local server was tested and the necessary data was transferred. Lastly the system as a whole was tested. All the specified sensors were created and the data designed to simulate the physical sensor were transferred at the desired time interval. All test were done manually which is why this segment is the sole documentation.

It can be concluded that the dummy sensor was working without any bugs.

## 5.2. Physical sensor test

To test the product we had to place it under the circumstances that it will have to endure in its deployment phase. The sensor we have on the prototype is a LM35 sensor attached to a chipset with wifi capabilities. When placing the sensor in a fridge we got reasonable measurements however the temperature range of the sensor were not as we had thought. Since we have a basic centigrade temperature sensor and not the full-range centigrade temperature sensor our range was from 2 °C to 150 °C (See appendix 1). This means we can not have the necessary measurements for a freezer. Being a prototype this is not our biggest concern but the sensor will obviously be replaced by a proper full-range sensor.

The signal receiver strength of the wifi chip did not pose any problems. This is necessary to consider because of the location of the chipset and sensor. So far it has to be within the fridge and therefore the wifi signal must be able to transmit through the fridge materials.

The primary purpose for the physical sensor was the provide proof of concept of the hardware part of the system. To fulfil this the following requirements needed to be satisfied.

1. The sensor must be able to connect to a local hotspot
2. Data must be transferred over a WiFi signal through a fridge and freezer
3. A reliable temperature must be measured
4. The local server must be able to receive the data sent
5. It must work at the temperature in a fridge and freezer

The physical sensor was tested manually. After the code was written and the sensor was assembled, it was tested at room temperature. The sensor was able to connect to a WiFi hotspot on an android, iphone and router. The sensor started sending data to the local server running on a laptop that was connected to the same hotspot. The temperature measured matched that of the control thermometer and responded in real time when subjected to heat and cold.

In a later test the sensor was placed in a fridge for 2 hours while on. The sensor worked as in the previous test. Then it was placed in a freezer for 2 hours. This gave the same result.

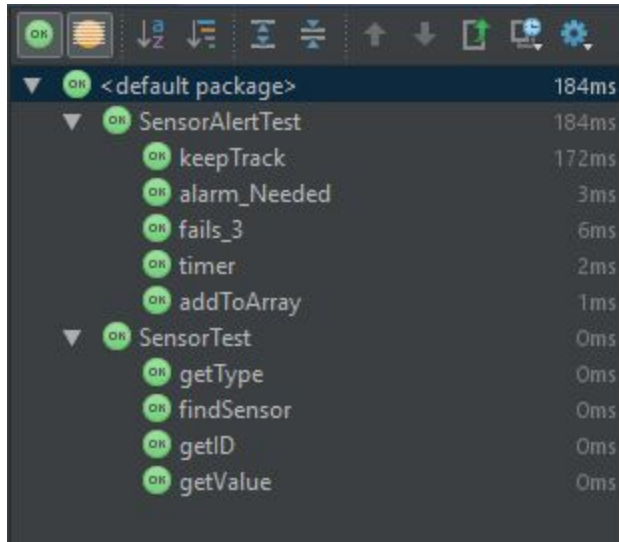
It was concluded that the sensor fulfilled the requirements of the proof of concept. Furthermore several important things were discovered during the creation and test of the sensor.

- A different temperatur sensor that have the range of -30 to 150 degrees celsius is needed
- A microprocessor that have a deep sleep function is needed to save power
- It is probably better to use AA batteries instead of a 9V
- Experience with arduino have show that it is a good idea to have hardware of the sensor behind an airtight seal

Before a market ready sensor can be created further research and development is needed.

### 5.3. Local server test

The classes SensorAlert and Sensor was tested by a black box junit test. All tests passed (see picture below).



INet and IDatabase are interfaces therefore they were not tested. The classes Sql and Tcq were tested indirectly by the database test, as they are used to connect to the database.

The remaining classes were tested when the whole system was tested. The following procedure was used.

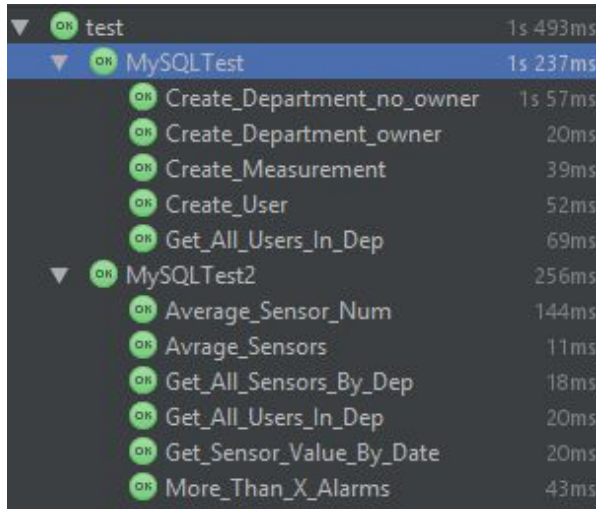
1. The database and web site was hosted on a pc of site.
2. The local server was started on a laptop then. The server prints what it does with the data received
3. The dummy sensor program was started on the same laptop. The dummy sensor prints the data that it sends to the local server
4. A browser was opened and we navigated to our web site ([http://dyrsted.duckdns.org:8888/CDIO\\_WebProject/](http://dyrsted.duckdns.org:8888/CDIO_WebProject/)) and logged in
5. A database management program was started on the same laptop and a connection to the database was established
6. The data was compared across all systems. Both measurements and alarms was found to be consistent.

It was concluded that the local server and the system as a whole worked and bug free.



## 5.4. Database test

To test our functions and procedures we have created a Java junit test which inserts data in the database using our functions and procedures and afterwards makes sure the data is inserted correctly. Using a view shows functions as a test as well since it's sole purpose is to access and show the data and since we have access to the tables directly we can simply check the tables and the view to test it's functionality.



test	1s 493ms
MySQLTest	1s 237ms
Create_Department_no_owner	1s 57ms
Create_Department_owner	20ms
Create_Measurement	39ms
Create_User	52ms
Get_All_Users_In_Dep	69ms
MySQLTest2	256ms
Average_Sensor_Num	144ms
Avrage_Sensors	11ms
Get_All_Sensors_By_Dep	18ms
Get_All_Users_In_Dep	20ms
Get_Sensor_Value_By_Date	20ms
More_Than_X_Alarms	43ms

Due to time restrictions recently added procedures have only been tested through a database management program, where we manually validated that the input matched the created tulip in the table(s). The new procedures are as follows.

- Create\_Alarm
- Create\_User
- Get\_Dept
- Remove\_Employee
- show\_alarms

It was concluded that the database functions to the desired specifications.

## 5.5. Website test

The website is easy to test since it requires nothing more than manual tests. Every Time a feature is implemented it simply has to be checked to see if the feature showed what was desired. The only features that could be tested in another way than visually would be the hosting of the website. But this can be tested by accessing the website from another device on another network.

## 6. Conclusion

Our system is capable of measuring the temperature of different kitchen appliances with an arduino attached to a breadboard equipped with a sensor which transmits data to our Java server. The Java server then determines whether or not the measurement is acceptable and then transmit data to the global database cloud. If it isn't it will send an additional value stating that an alarm must be triggered. In the end product it would also have to make a physical alarm go off but at the end of this assignment it is simply a signal to the database. Our front-end website neatly shows and updates the measurements taken, and can effortlessly handle and display several measurements from our dummy sensors. The website is also used to add, edit and delete users in the system. All Data is stored in our database but can only be viewed using appropriate credentials on the website since the access has been limited to allow specific people different information. The process was smooth and went without much trouble since most of the concept was already in place at the end of CDIO 3. All the requirements were met and the assignment was satisfied.

# 7. Appendix

## 7.1. Appendix 1



Product  
Folder



Sample &  
Buy



Technical  
Documents



Tools &  
Software



Support &  
Community



LM35

SNIS159G – AUGUST 1999 – REVISED AUGUST 2016

### LM35 Precision Centigrade Temperature Sensors

#### 1 Features

- Calibrated Directly in Celsius (Centigrade)
- Linear + 10-mV/°C Scale Factor
- 0.5°C Ensured Accuracy (at 25°C)
- Rated for Full -55°C to 150°C Range
- Suitable for Remote Applications
- Low-Cost Due to Wafer-Level Trimming
- Operates from 4 V to 30 V
- Less than 60-μA Current Drain
- Low Self-Heating, 0.08°C in Still Air
- Non-Linearity Only ±¼°C Typical
- Low-Impedance Output, 0.1 Ω for 1-mA Load

#### 2 Applications

- Power Supplies
- Battery Management
- HVAC
- Appliances

#### 3 Description

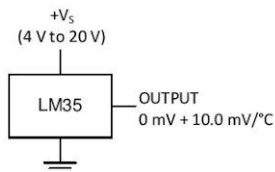
The LM35 series are precision integrated-circuit temperature devices with an output voltage linearly-proportional to the Centigrade temperature. The LM35 device has an advantage over linear temperature sensors calibrated in Kelvin, as the user is not required to subtract a large constant voltage from the output to obtain convenient Centigrade scaling. The LM35 device does not require any external calibration or trimming to provide typical accuracies of ±¼°C at room temperature and ±¾°C over a full -55°C to 150°C temperature range. Lower cost is assured by trimming and calibration at the wafer level. The low-output impedance, linear output, and precise inherent calibration of the LM35 device makes interfacing to readout or control circuitry especially easy. The device is used with single power supplies, or with plus and minus supplies. As the LM35 device draws only 60 μA from the supply, it has very low self-heating of less than 0.1°C in still air. The LM35 device is rated to operate over a -55°C to 150°C temperature range, while the LM35C device is rated for a -40°C to 110°C range (-10° with improved accuracy). The LM35-series devices are available packaged in hermetic TO transistor packages, while the LM35C, LM35CA, and LM35D devices are available in the plastic TO-92 transistor package. The LM35D device is available in an 8-lead surface-mount small-outline package and a plastic TO-220 package.

#### Device Information<sup>(1)</sup>

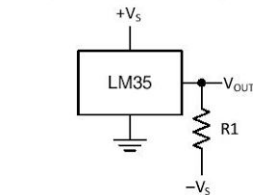
PART NUMBER	PACKAGE	BODY SIZE (NOM)
LM35	TO-CAN (3)	4.699 mm × 4.699 mm
	TO-92 (3)	4.30 mm × 4.30 mm
	SOIC (8)	4.90 mm × 3.91 mm
	TO-220 (3)	14.986 mm × 10.16 mm

(1) For all available packages, see the orderable addendum at the end of the datasheet.

#### Basic Centigrade Temperature Sensor (2°C to 150°C)



#### Full-Range Centigrade Temperature Sensor



Choose  $R_1 = -V_S / 50 \mu\text{A}$   
 $V_{OUT} = 1500 \text{ mV at } 150^\circ\text{C}$   
 $V_{OUT} = 250 \text{ mV at } 25^\circ\text{C}$   
 $V_{OUT} = -550 \text{ mV at } -55^\circ\text{C}$



An IMPORTANT NOTICE at the end of this data sheet addresses availability, warranty, changes, use in safety-critical applications, intellectual property matters and other important disclaimers. PRODUCTION DATA.