**CST-361 Design Patterns in Java**

**Project Title: Milestone 1.**

Group 2

Date: May 24th, 2020

**CST-361 - Design Report**

|  |  |  |
| --- | --- | --- |
| **Topic:** | ***Weather Detection Application: Milestone 1*** | |
| **Date:** | *24/05/2020* | |
| **Revision:** | *1.0* | |
| **Team:** | 1. Jake Taylor | |
| 1. Raymond Lawson | |
| 1. Vien Nguyen | |
| **Weekly Team Status Summary:** | |  |  |  |  | | --- | --- | --- | --- | | **User Story** | **Team**  **Member** | **Hours**  **Worked** | **Hours Remaining** | | Scrum Planning | Jake | 2hrs | 0hr | | Milestone Delivery Calendar | Jake | 1hr | 0hr | | UML Class Diagram for a Potential IoT device that is capturing data | Jake | 1hr | 0hr | | UML Deployment Diagram | Jake | 1hr | 0hr | | UML Case Diagram for IoT Device & End User | Raymond | 1hr | 0hr | | Risk and Unknown | Raymond | 1hr | 0hr | | Deployment Strategy for IoT Device | Vien | 1hr | 2hrs | | UML Component Diagram Depicting Logical Block Diagram for IoT Solution | Vien | 1hr | 0hr | | Research existing IoT applications | Vien | 2hrs | 3hrs | | General Teach Approach | Vien | 1hr | 0hr | | |
| **GIT URL:** | [*https://github.com/btzblackout/CST-361-CLC-Project*](https://github.com/btzblackout/CST-361-CLC-Project) | |
| **Peer Review:** | *Y/N* | We acknowledge that our team has reviewed this report and we agree to the approach we are all taking. |

**Planning Documentation**

**Scrum Planning**

* Team Organization:
  + Product Owners - Jake Taylor, Vien Nguyen, Raymond Lawson
  + Scrum Master - Jake Taylor
  + Agile Team - Jake Taylor, Vien Nguyen, Raymond Lawson
* Team Meetings:
  + Team will meet through Discord on an “as needed” basis throughout each week. Collaboration on documents will be done through Microsoft Word and Discord.
* Peer Reviews:
  + Topics: What went well, what didn’t go well, action plan for each
  + When: Start of each week after submission of Milestone
* Project Requirements:
  + Cover Sheet - to include the approach that will be taken to complete project
  + Technical Elements -
    - UML Diagrams, deployment strategy, wireframes, Java EE Code, Raspberry Pi, REST API, MySQL DB, Front-End Development
  + Non-Technical Elements -
    - Planning & delivery methodology, risks and unknowns, IoT data, IoT device
* Project Components:
  + Jake: Scrum Planning, Code Development, Project Delivery
  + Raymond: Code Development, Project Delivery
  + Vien: Code Development, Project Delivery

**Milestone Delivery Calendar:**

*May*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Sun | Mon | Tues | Wed | Thurs | Fri | Sat |
| 26 | 27 | 28 | 29 | 30 | 1 | 2 |
| 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 10 | 11 | 12 | 13 | 14 | 15 | 16 |
| 17 | 18 | 19 | 20 | 21 | 22 | 23 |
| 24  Milestone 1 & 2 | 25 | 26 | 27 | 28 | 29 | 30 |
| 31 | 1 | 2 | 3 | 4 | 5 | 6 |

*June*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Sun | Mon | Tues | Wed | Thurs | Fri | Sat |
| 31 | 1 | 2 | 3 | 4 | 5 | 6 |
| 7  Milestone 3 | 8 | 9 | 10 | 11 | 12 | 13 |
| 14  Milestone 4 | 15 | 16 | 17 | 18 | 19 | 20 |
| 21 | 22 | 23 | 24 | 25 | 26 | 27 |
| 28 | 29 | 30 | 1 | 2 | 3 | 4 |
| 5 | 6 | 7 | 8 | 9 | 10 | 11 |

*July*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Sun | Mon | Tues | Wed | Thurs | Fri | Sat |
| 28 | 29 | 30 | 1 | 2 | 3 | 4 |
| 5  Milestone 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| 12  Milestone 6 & 7 | 13 | 14 | 15 | 16 | 17 | 18 |
| 19 | 20 | 21 | 22 | 23 | 24 | 25 |
| 26 | 27 | 28 | 29 | 30 | 31 | 1 |
| 2 | 3 | 4 | 5 | 6 | 7 | 8 |

**Agile Scrum Product Backlog:**

*This needs to contain a URL to Bitbucket Scrum Product Backlog Artifact.*

**Agile Scrum Sprint Backlog:**

*This needs to contain a URL to Bitbucket Scrum Sprint Backlog Artifact. This current week’s progress should be reflected in the above section of this Design Report.*

**Agile Scrum Burn Down Chart:**

*This needs to contain a URL to Bitbucket Scrum Burn Down Chart Artifact.*

**Agile Retrospective Results:**

*The following table should be completed after each Retrospective on Things That Went Well (Keep Doing). An alternative to the following table is to use a Mind Mapping tool such as Coggle. If you use a Mind Mapping tool you must include a URL or Image File.*

|  |
| --- |
| **What Went Well** |
| Team members are excellent at communication. Each member was assigned and finished their own tasks. |
|  |
|  |

*The following table should be completed after each Retrospective on Things That Didn’t Go Well (Stop Doing) and What Would Be Done Differently Next Time with an Action Plan to Improve (Try Doing and Continuous Improvement). An alternative to the following table is to use a Mind Mapping tool such as Coggle. If you use a Mind Mapping tool, you must include a URL or Image File.*

|  |  |  |
| --- | --- | --- |
| **What Did Not Go Well** | **Action Plan** | **Due Date** |
|  |  |  |
|  |  |  |
|  |  |  |

**Design Documentation**

**Install Instructions:**

*N/A*

**General Technical Approach:**

The idea of weather detection application is the corporation of hardware and software, and is divided into different implementation levels.

**Front end web application**: the weather app will be created on Enterprise Java to provide the user interfaces and to show the weather data (could be a real-time data collected from the sensors which is sent from the back end). This level is implemented by JSF, CSS, JavaScript, charting libraries.

**Back end service**: the weather app is using Enterprise Java to build the RestFul-services so that it can communicate with the front-end, devices, database server. Team members will research to implement java code, database.

**Embedding:** this embedding is going to be an alternative replacement for sensors to issue data (real devices could be used).

This project is collaborating work, so the project code management could be used to save and share between members. Github is the option so far.

**Key Technical Design Decisions:**

*Any final technical design decisions, such as framework decisions etc., should be documented here. This should list the technology/framework, its purpose in the design, and why it was chosen.*

**Known Issues:**

*Any anomalies or known issues in the code or functionality should be documented here.*

**Risks:**

One risk is the security of information passed between nodes and it is remedied with encryption.

**Unknown:**

Some unknowns at the moment are implementing the connection from the raspberry pi to the server, and from the database to the server.

**ER Diagram:**

*N/A*

**DDL Scripts:**

*N/A*

**Flow Charts:**

*N/A*

**Sitemap Diagram:**

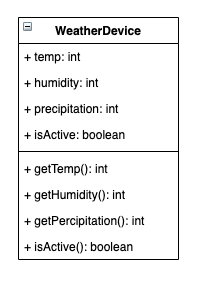
*N/A*

**User Interface Diagrams:**

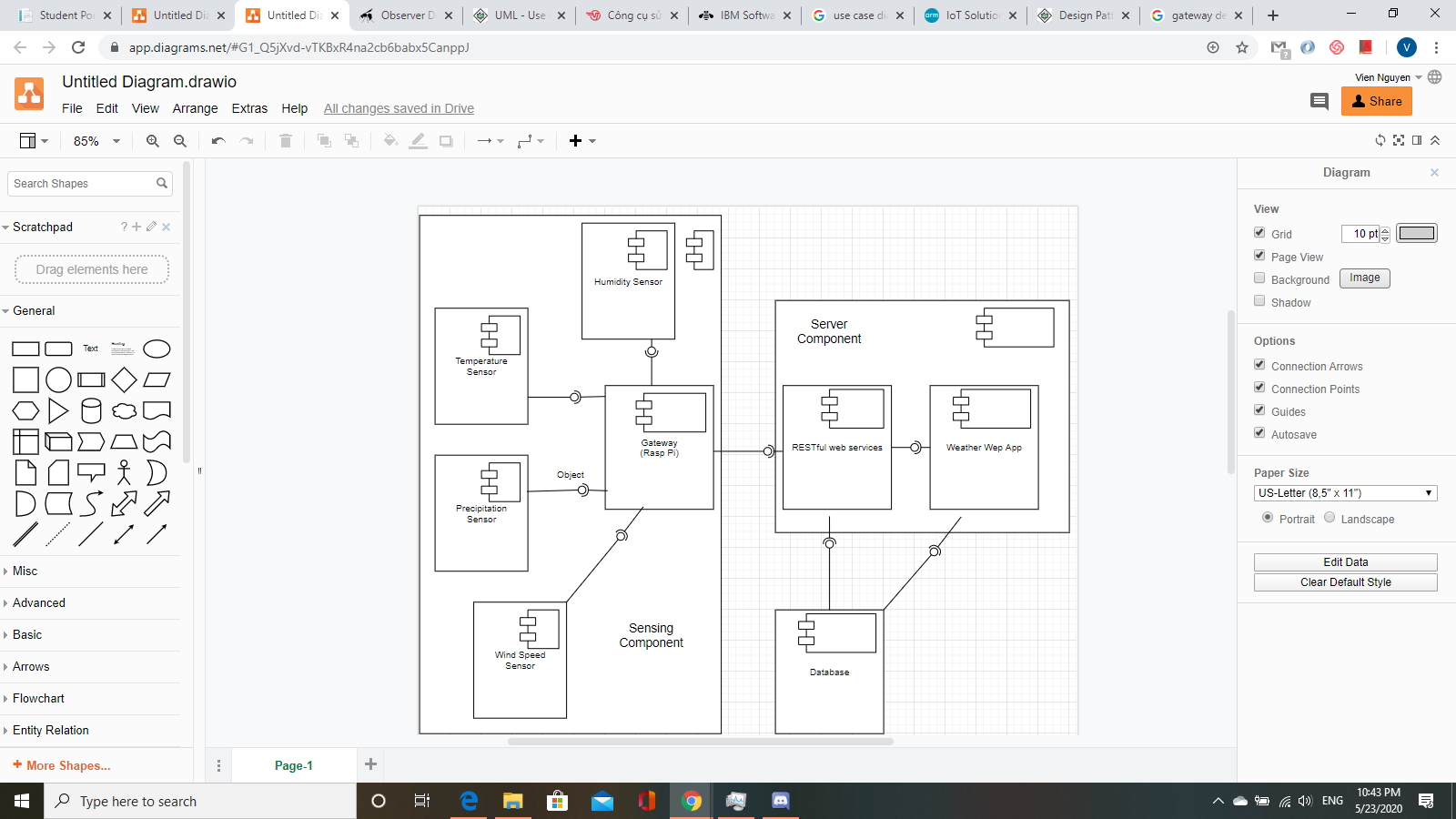
*N/A.*

**Class Diagrams:**

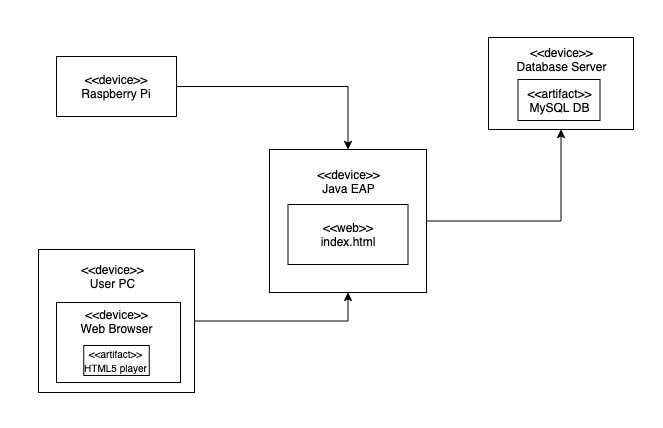
**UML Class Diagram for a Potential IoT device that is capturing data**

****

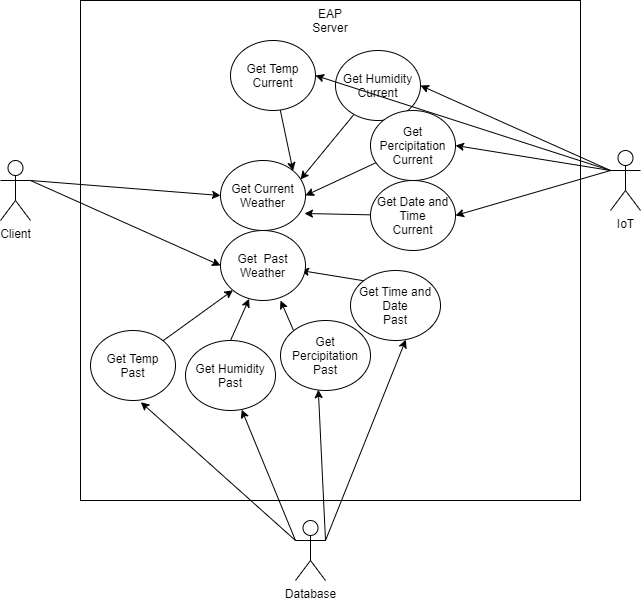
**UML Component Diagram Depicting Logical Block Diagram for IoT Solution:**

****

**UML Deployment Diagram:**

****

**UML Case Diagram for IoT Device & End User:**

****

**Service API Design:**

*N/A*

**Security Design:**

*N/A*

**Pseudo Code:**

*N/A.*

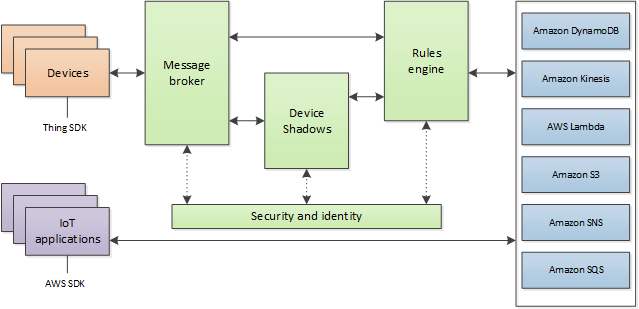
**Other Documentation:**

*N/A.*

**Research existing IoT applications.**

**Alexa Voice Service (AWS):**

AWS is a system that allows users to interact with the home IoT devices. AWS plays a role of cloud computing that listens to instruction of users by voices. Home IoT devices are connected to AWS cloud through wifi or cable and are identified in the network. Whenever AWS receives a request from a user, the server will generate a compatible instruction message into the channel. If the devices have been subscribed to that channel, it can receive the control from the system. The below figure is the simple architecture of AWS IoT.



**Deployment Strategy for IoT Device:**

The Weather Detection Application is a system of devices and software that allows weather data to be collected from the sensors and sent to the cloud server. The weather application will be a simple system of at least one or two Raspberry Pi, and computer.

Raspberry Pi is installed on the Raspbian OS, and Docker software so that it can run different programming languages and software in docker containers. At this stage, the sensors are simulated by java code and run in Java docker to create data like temperature, humidity, wind speed, location, etc. Raspberry Pi also works as a gateway to communicate with the server.

A server will have the database management installed in and a weather detection application. This application receives data sent from Raspberry Pi and saves it to the database.

Client PC can access or retrieve weather data from the java web application.