CA326 Technical Manual/Specification

- **Project Title:** Air Quality Monitor via Raspberry Pi
- **Student Names:** Ronghui Lin **Student ID:** 19354553
  Vaidas Buzas **Student ID:** 19322426
- **Supervisor:** Renaat Verbruggen

Table of Contents

1 Introduction

Air Quality Monitor via Raspberry Pi is a real time web application that displays information from sensors connected to a Raspberry Pi, allowing real time monitoring of indoor or outdoor air quality data. The web app itself is built with Node.js and Express for server side handling while the front end consisted of HTML, EJS and Bootstrap for styling. It utilises SQLite3 for database handling.
Although it is recommended that we move to a more robust database management system as SQLite3 is more suited for small projects. It is targeted at the sustainability market, although it isn’t necessarily only built for that market, it can be used for other cases as well.

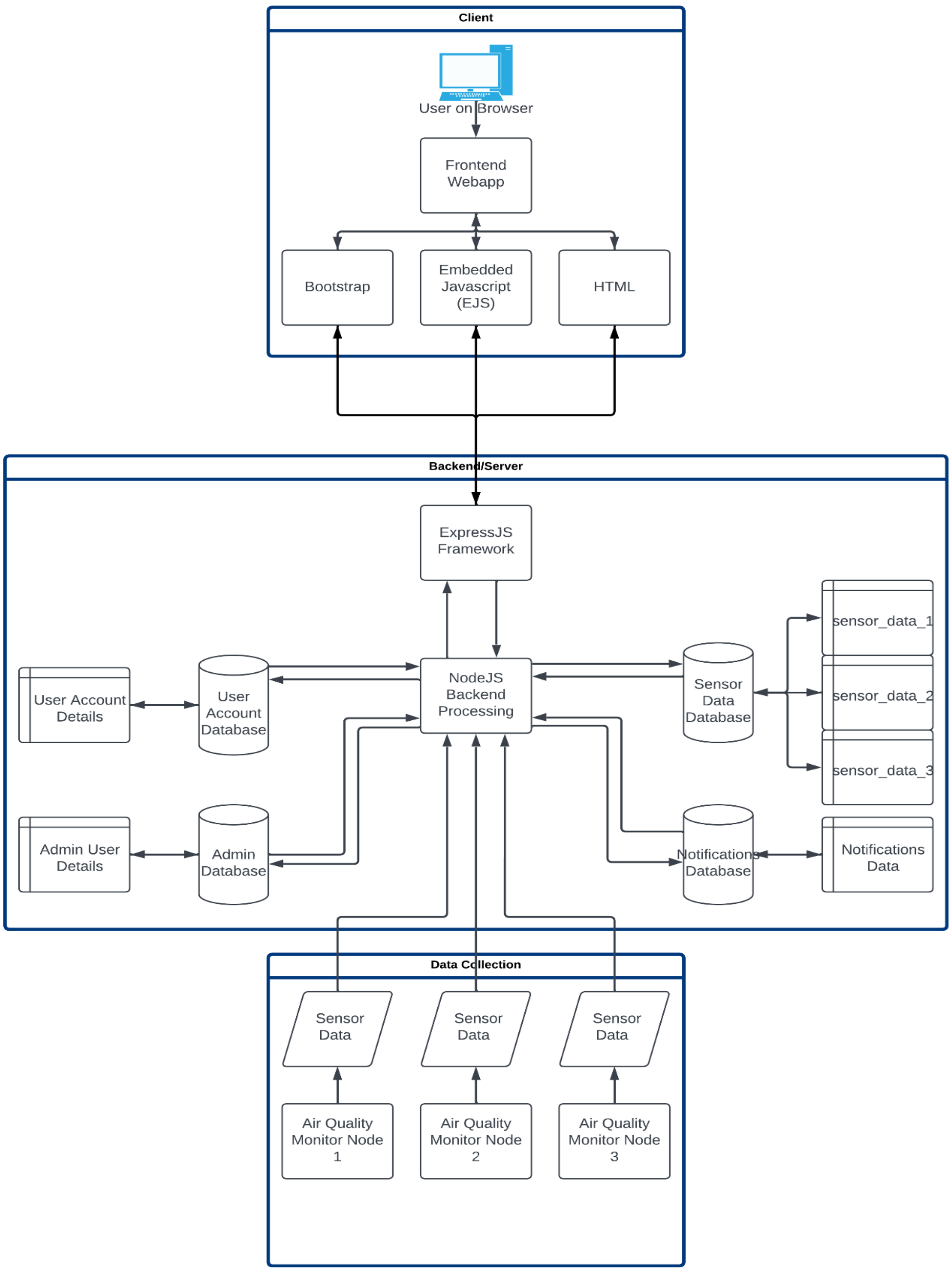
1.2 Glossary

Term	**Definition**
UI	User Interface
JSON	Javascript Object Notation, a file format for data interchange
EJS	Embedded JavaScript
SMTP	Simple Mail Transfer Protocol, used by servers to send, receive or relay outgoing mail.
CSV	Comma-Separated Values file is a text file that uses a comma to separate values.
CPU	Central Processing Unit, the brain of the computer that does all the calculations.
Frontend	What is displayed to the user on a website.
Backend	Where all the processing and

	calculations occur behind the scenes.
Middleware	Functions in NodeJSthat have access to request and response objects
Bootstrap	CSS framework for responsive front end development

2 System Architecture

2.1 Architecture Diagram:



2.2 Embedded Javascript (EJS)

For our templating, we’re using EJS to handle all the templates, different views for pages and management of the head, header and footers. This is to ensure uniformity across the web application so there is consistency across the site.

2.3 HTML & Bootstrap

Our frontend website is written in HTML and styled with Bootstrap. Users when arriving at the site will be greeted by our site and this is where the main user interface appears. Unregistered users will be able to view the sensor data available, they may choose to login or register for an account.
Once registered users login, they will be able to create notifications to monitor their selected node and their inputted thresholds for the data they want to monitor. They will also be able to export all the data out of the database for the specific node they have chosen to .CSV format.
Administrator users will be able to manage notifications and sensor data on top of what registered users are able to do.

2.4 ExpressJS Web Framework

ExpressJS is used in our application in order to provide us with the ability to use middleware, routing and templating in addition to EJS. This framework speeds up the development time and allows us to link the frontend and backend processing. Another crucial component was that express allows us to create dynamic rendering of HTML pages, which is vital for showing multiple nodes dynamically as we add more nodes to the system.

2.5 NodeJS

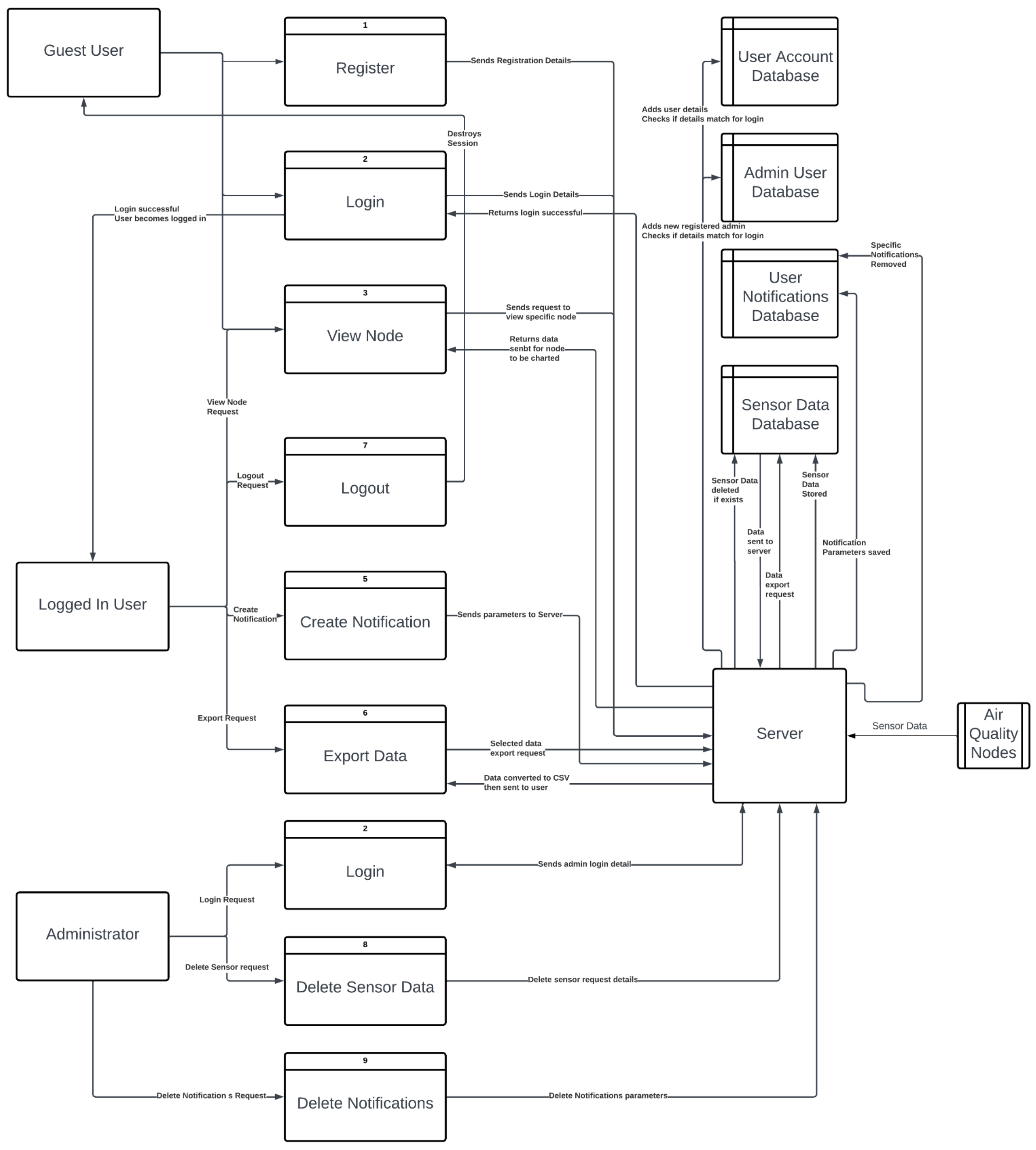
NodeJS is the server that handles all the backend processing of data and requests, this is where the main functionality of the application is programmed. It uses SQLite3 to store the sensor data delivered into their own databases and also manages three other tables which include admin users, regular users and email notifications. Once the NodeJS backend receives a request, it sends a response usually in JSON form to allow the frontend to read it and display it to the user.
It also handles all the email notifications, sending emails through SMTP through the use of nodemailer. User passwords are also encrypted through bcrypt’s hashing and salting, so passwords will be securely stored in the database.

2.6 Data Collection Layer

The data collection layer consists of the Raspberry Pi nodes with their sensors connected.
The program to collect the data from the sensors and package them into a dictionary is written in python and is then sent to the NodeJS backend where it is stored according to the name set for the payload. Calibration of the data collected is also processed on the Pi, for example the compensated CPU temperature, this is required as the temperature sensor is close to the CPU, as such a maths calculation is required to take into account the current CPU temperature.

3. High-Level Design

3.1 Data Flow Diagram



In the Data Flow Diagram above, it shows how the user would interact with the frontend through a browser. It then shows systems interacting with one another when a user interacts with it. This also shows the availability features that are open depending on the level of access the user has, as only an administrator has access to to deletion and regular logged in users can have access to export and notifications. In this it also shows how data or requests flow to another component of the system and how it handles it and sends it on.

3.2 Language Choices

Javascript was chosen to be the main language as upon research, NodeJS has a great library to utilise due to the default package manager being npm which allows us to import modules to increase how aesthetically pleasing the application will look as well as other modules that support the development of the web application more efficiently. Its support of ExpressJS also proved pivotal for us as it allowed for one language to be used for the frontend, middleware and backend which reduces development time and issues with multiple languages.

Python 3 is also used, however its use is limited to data collection on the Raspberry Pi4, it was primarily used due to our familiarity with the language and also some of the sensors have libraries built for them in python already, which contributed massively to the reason why we used it for the data collection.

4. Problems and Resolution

4.1 Problem 1 - Multi-node Support

Many problems in this part of the project can be attributed to poor planning, which will become a running theme. One of which was the system initially was hardcoded to show only one sensor node in order to test functionality, however as more features were added, functions became interdependent on each other and by the time we realised that the web app was hardcoded to one single node and cannot accept another node, it was too complex to be changed easily. As a result of this, we rewrote parts of the system, such as the /store endpoint in order to accept different nodes and store it in different tables. This by nature extended to various parts of the web application, not just the data storage.

4.2 Problem 2 - Dynamic Node View Selection

This problem can also be attributed to problem 1, dynamic node viewing is where we have the system dynamically grab the table names from the database and add it to a drop down menu, allowing users to see new nodes available to view as they connect, instead of us manually creating a new view in the pages directory, the system will dynamically create it.
Unfortunately, due to hard coding in problem 1, this proved to be difficult and it required a rewrite of the data.ejs file, a major rewrite of the navigational bar and required us to write multiple scripts in order to get it functional. Once it was updated and functional, replicating it became much easier and the same dynamic process was used for the /export function later.

4.3 Problem 3 - Real Time Chart Updates

The charts throughout most of the development process were functioning and were updating with data sent from the raspberry pi to the server, however new data was only being displayed once the users refresh the page. This was a serious problem as at that point, due to the way we have the routes programmed and how the URLs were laid out, there was no way for the /data endpoint to request the server to send updated values back to the frontend without refreshing the page.
Solution to this problem required again, a major modification to the data.js file and the creation of a new middleware function named /chartUpdate. In order to get the tableName variable, processing of the URL was achieved through another function called getParameterByName which would grab it once the page loads.

5. Testing

5.1 Unit Testing Python (PyUnit)

With the data collection on the raspberry pi being written in python, we implemented unit tests for each function and the data transmission function with PyUnit. The tests are written to ensure that each function and sensor is collecting information correctly, the data transmission test is used to check if the data posted to the server is formatted correctly in JSON.

5.2 Unit Test Javascript (Mocha)

On the server side, as Javascript is being used through NodeJS and Express, we decided on Mocha as it can test both server side and client side and is a test framework designed for NodeJS. The unit tests here checks if /sensor-data-tables returns with a JSON array of the sensor tables in the database. It also checks multiple routes and middleware to see if it responds with data correctly and checks if the response is correct if a user is logged in, not logged in or is/isn’t an admin.

These were essential as they were core parts of the system for the entire application to work and run cohesively.

6. Installation Guide

5.1 Download and Installation

Please install NodeJS first HERE,
After installation, open up Command Line and run “npm install DEPENDENCY”
Replace DEPENDENCY with each dependency listed below:
Dependencies Server Side:

- ExpressJS
- npm
- SQLite3
- Express Session
- BCrypt
- Dotenv
- Nodemailer
- EJS
- NodeJS
- Bootstrap
- JQuery
- Chartjs
- Chartjs Annotation

Dependencies Data Collection side (Raspberry Pi)

- Clone the github repository
- https://gitlab.computing.dcu.ie/linr2/2023-ca326-ThirdYearProject
- Run install.sh
  This should install most if not all dependencies.

Clone the repository to the device you want to act as server.
Then,

1. Change Directory to the Website directory
2. Type and run this command, “node server.js”
3. Server should be up and running.

On the Raspberry Pi end,

1. Make sure you’re using the latest stable Raspberry Pi OS.
2. Once the repo has been cloned, change directory to the root directory where start.py is located.
3. Run this command “python3 start.py”.