Energy Insight Hub Project Report

by Alex Paquette

Student Number: C00302989

Supervisor: Dr. Oisin Cawley

Submitted To: South East Technological University

Date: April 19th, 2024

Abstract

The “Energy Insight Hub” project endeavours to develop a comprehensive software and hardware solution for energy meter reading. This eight-month-long endeavour has been guided by the aim of creating an efficient system capable of extracting readings from meter images and presenting energy consumption data to users via a front-end web application.

Throughout the development process, various technical challenges were encountered, including hardware integration difficulties, software compatibility issues, and data processing complexities. Despite these hurdles, progress was made in achieving key project objectives, including the development of a functional front-end interface and the successful extraction of meter readings from images.

This abstract encapsulates the project’s journey, outlining the primary objectives, challenges faced, and achievements made. By reflecting on these experiences, insights into the process of software development and hardware integration are gained.

Acknowledgements

I would like to express my sincere gratitude to Dr. Oisin Cawley for his guidance, support, and mentorship throughout the duration of this project. His expertise and constructive feedback have been instrumental in shaping the direction of the Energy Insight Hub project.

I would also like to extend my appreciation to the faculty and staff of South East Technological University for their support and resources provided during the process.

Contents

Introduction

The “Energy Insight Hub” project represents a culmination of efforts aimed at developing a comprehensive software and hardware solution for remote energy meter reading. Over the course of the eight-month-long endeavour, various technical and logistical challenges were encountered, leading to invaluable learning experiences and insights into software development and hardware integration.

The primary objective of this project was to create an energy meter reader system comprising hardware and software components. Key components included a front-end web application for data visualization, a script for extracting readings from meter images, and hardware integration for real-world data capture. Challenges ranged from hardware integration difficulties to software compatibility issues.

This report aims to provide a comprehensive overview of the Energy Insight Hub project, detailing challenges encountered, achievements made, unachieved goals, and lessons learned. By reflecting on these experiences, valuable insights into the process of software development and hardware integration are gleaned.

General Issues

Problems Encountered

Seven-Segment OCR Challenges

Performing optical character recognition (OCR) on seven-segment digits from meter images posed insurmountable challenges with existing OCR libraries. Contrary to initial assumptions, conventional OCR libraries, such as pytesseract in Python, were unable to recognize or extract digits from seven-segment displays. Figure 1 illustrates an image of a modified meter display with seven-segment digits, where conventional OCR methods failed to yield results.

Figure 1. Image of a modified meter display showing seven-segment digits.

However, success was achieved in extracting digits from artificial meter images, as seen in Figure 2.

Figure 2. Artificial image of an energy meter, modified by Alex Paquette. (Author unknown, from blacklionhvac.com)

To address this limitation, an extensive investigation into alternate solutions was conducted. Subsequently, a packaged CLI Linux application developed by Erik Auerswald proved a viable solution for extracting digits from seven-segment displays. The approach involved invoking the LCI command from a Python method and capturing its output to generate the reading. Refer to the documentation for detailed implementation steps under the method `read_meter_old(meterPhoto)`.

Web App Hosting Difficulty

Hosting the web application on a Linux environment presented unexpected challenges, resulting in compatibility and functionality issues. Initially developed and tested in a Windows environment, the transition to hosting on Linux became necessary to address the SSOCR problem. Despite efforts to adhere to Microsoft's guidelines for Linux hosting, the server

interactivity component of the web app remained non-functional on the Linux platform. The root cause of this issue remains unknown, but it is suspected to be related to missing libraries required for enabling this functionality within the operating system.

Furthermore, the interaction between the web app and the Windows Subsystem for Linux (WSL) environment may have contributed to the observed discrepancies compared to a native Linux environment. Regrettably, due to time constraints, a comprehensive resolution to this challenge could not be achieved, and the web app is currently hosted on Windows.

The inability to resolve this hosting issue had a cascading effect, as the solution devised for the SSOCR problem could not be implemented in the final version of the project. This limitation underscores the importance of seamless web app hosting for the successful integration of project components.

Database Write Detection Issues

Difficulty in detecting when the database was written posed challenges in ensuring real-time updates of the live dashboard component in the front-end application. The primary issue stemmed from the database being locked during write operations, preventing the `FileSystemWatcher` from detecting changes in the database.

To address this challenge, a workaround was devised by implementing an auxiliary file, update.txt, which served as a signalling mechanism for database updates. Whenever a new reading was added to the database, the update.txt file was concurrently written to. This approach enabled the `FileSystemWatcher` to detect write operations to the update.txt file, signalling the occurrence of database updates.

The implementation of this solution is encapsulated in the constructor of the `DashboardUpdate` class. By leveraging this mechanism, real-time updates of the live dashboard component were achieved, ensuring the display of the most recent data to users.

Multiple Processes of Meter Reader

The occurrence of multiple processes of the meter reader running subsequently presented challenges in data integrity and system stability. Initially, the web app was modified to initiate a process of the MeterReader Python script upon startup. However, a critical oversight led to the MeterReader process persisting beyond the termination of the web app, resulting in the simultaneous execution of multiple instances of the process and subsequent duplication of entries in the database.

To mitigate this issue, a solution was devised by capturing the process ID (PID) and saving it in a text file. Upon creation of the process, the PID was recorded, enabling subsequent startups to check for the existence of a lingering MeterReader process, and terminating it before starting a new instance.

The solution to terminate the lingering process was adapted from a post on Stack Overflow by user SulNR. Integration of this solution was incorporated into the `EndService(int pid)` method of the `MeterReaderService` class.

Hardware Integration Challenges

The development of the Energy Insight Hub project encountered hurdles in wiring a camera to the microcontroller for real-world image capture. Integrating the hardware components proved to be significantly more complex than initially anticipated.

To facilitate the integration process, reference was made to the guide “How to Use OV7670 Camera Module with Arduino Uno”. This resource provided valuable insights into the wiring configuration required for connecting the camera to the microcontroller. Figure 3 illustrates a circuit diagram depicting the interconnection between the Arduino UNO and the OV7670 camera module, serving as a guide for the wiring process.

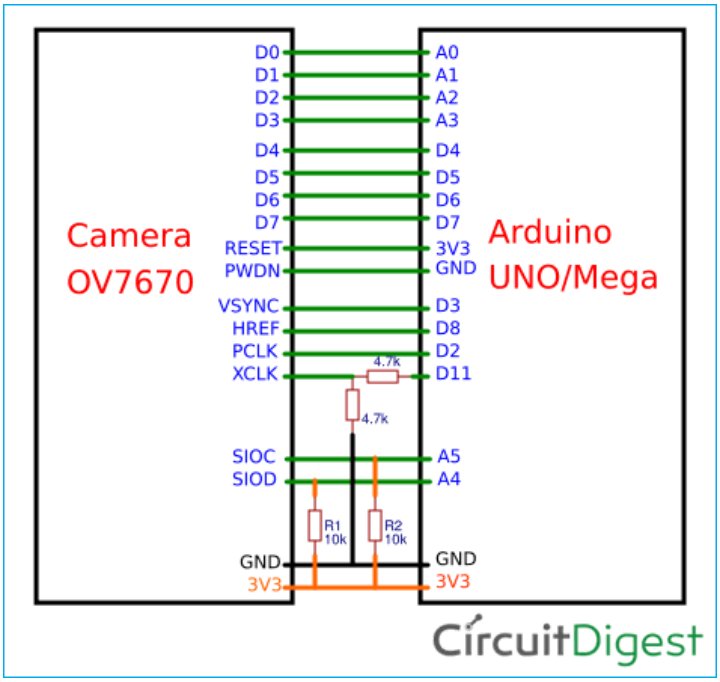

Figure 3. Circuit Diagram between Arduino UNO and Camera OV7670. Shows how to wire components together. (Circuitdigest.com, 2019).

Despite earnest efforts to adhere to the circuit diagram, challenges arose due to a discrepancy in the grounding pins between the available hardware and the diagram. Specifically, the diagram depicted two 4.7k ground and two 10k ground, which were not readily available during the writing process. This mismatch impeded the successful completion of the hardware integration task, compounded by time constraints.

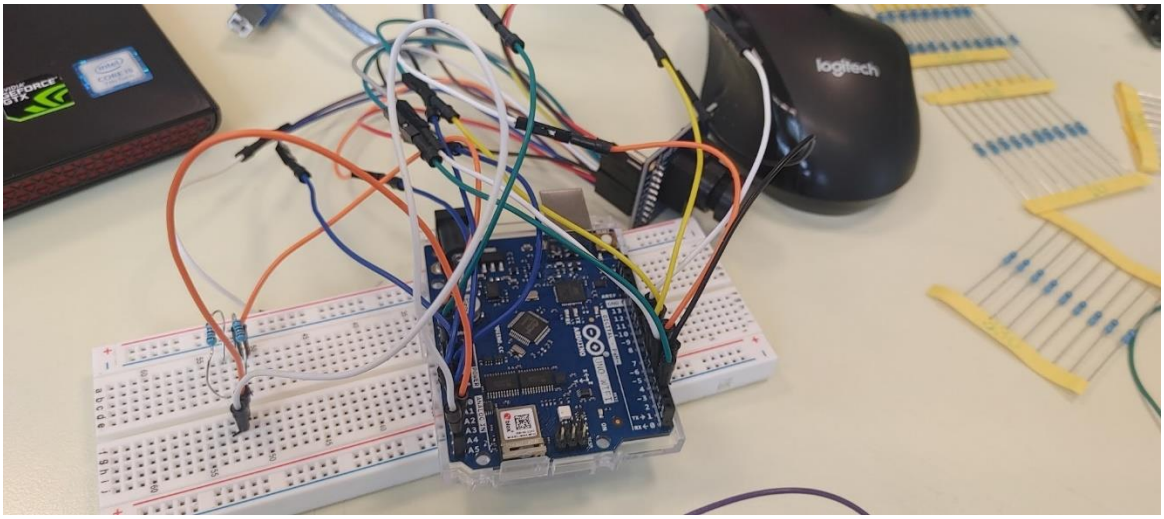

Figure 4. A partially wired Camera OV7670 to an Arduino UNO. Photographed by Alex Paquette March 19th 2024

Transmitting Data from the Microcontroller

Establishing a connection between the microcontroller and the server for data transmission proved to be challenging, particularly stemming from limitations in configuring network settings. Reference was made to the guide "*Add WiFi to Arduino UNO*" by Jeff Paredes to facilitate the connection process. Initially, attempts were made to connect the microcontroller to the college network. Although successful in the initial connection and pinging Google's IP address, subsequent reconnections failed. This is likely due to how the college network is configured, and lack of access prevented further investigation.

A workaround solution was found by using a personal hotspot for connectivity, which enabled the microcontroller to establish a stable internet connection and successfully conduct pinging activities.

Endeavours were made to establish a connection between the microcontroller and a personal computer but encountered difficulties. It was initially speculated that the microcontroller's inability to establish a connection with the computer was due to the firewall

blocking incoming ping requests. Following guidance from the article "Enable Ping Windows 10: How to Set Up Firewall to Allow It" by Henderson Jayden Harper, steps were taken to configure the personal computer to allow ping requests from external devices. However, despite meticulous adherence to the outlined steps, the issue persisted.

Time constraints prevented a resolution to this problem. The inability to establish a connection may be attributed to the network masking the personal computer’s IP address, preventing its discovery by the ping from the microcontroller.

Achievements

Functional Front-End Interface

A key achievement of the Energy Insight Hub was the development of a functional front-end interface. This interface features a live dashboard that enables users to view data consumption live as it is processed.

Meter Reading Extraction

Another significant accomplishment was the successful implementation of the meter reading extraction functionality. Leveraging the pre-packaged SSOCR (Seven-Segment Optical Character Recognition) solution developed by Erik Auerswald, the system achieved the ability to extract and process meter readings from images.

Unachieved Goals

Hardware Solution for Image Transmission

Despite efforts to integrate a hardware solution for image transmission from the meter reader to the server, unresolved challenges in hardware integration impeded the implementation of this feature. As a result, the project fell short of achieving its objective of deploying a fully functional hardware solution for image transmission. The complexity and scope of the hardware integration task proved insurmountable within the project timeline.

Lessons Learned

Importance of Hardware Knowledge

This project underscores the importance of having a thorough understanding of hardware components for successful integration into software systems. The unfamiliarity with the hardware technologies to be used in this project resulted in an incomplete system and the lack of any hardware component in the final deliverable. The challenges encountered in hardware integration highlight a need for greater expertise and resources in this area. More time during development should have been dedicated to properly understanding, developing, and integrating the microcontroller into the system.

Technical Issues

Deviations from Initial Design and Hardware Integration Challenges

The development of the project encountered significant deviations from the initial design, primarily stemming from challenges in hardware integration and limitations in hardware knowledge. The project's original design, including proposed class diagrams and hardware components, underwent substantial modifications due to encountered technical hurdles.

A notable challenge arose from the limited familiarity with hardware technologies and components. Despite efforts to configure the hardware components, such as the camera module and microcontroller, gaps in hardware knowledge impeded progress and led to the exclusion of hardware features from the final deliverable.

The combined impact of deviations from the initial design and challenges in hardware integration underscored the importance of possessing comprehensive knowledge of hardware technologies for successful software-hardware integration projects.

Module Descriptions

Radzen Components for Blazor

Radzen is a component library specifically designed for Blazor. It offers a wide range of pre-built components for constructing user interfaces. These components include layout elements, charts, buttons, tables, and text elements, among others. In this project, Radzen components were utilized to streamline the development of the front-end user interface, enabling the efficient creation of interactive and visually appealing dashboard elements.

SignalR for Real-Time Updates

SignalR is a real-time communication library in ASP.NET that facilitates bi-directional communication between server and client. In this project, SignalR was employed to enable live updates on the dashboard built with Radzen components. By establishing a connection between the server and the client, SignalR ensured that the dashboard would automatically refresh whenever new data was written to the database, providing users with real-time data visualizations of their energy consumption.

Python Scripting for Meter Reading Extraction

Python served as the scripting language for developing the module responsible for extracting meter readings from images. Utilizing libraries such as cv2 for image manipulation, os for file access, and SQLite3 for database interaction, the Python script processed meter images, extracted readings, and stored them in the database. Additionally, subprocess was used to execute Linux commands, and pytesseract was initially explored for OCR functionality before adopting SSOCR.

Entity Framework for Database Management

Entity Framework is a widely used Object-Relational Mapping (ORM) framework in the .NET ecosystem. It was employed to manage the project's database operations from the side of the web app. By leveraging Entity Framework, the database models were defined in C# by seamlessly scaffolding the database schema to the models. This streamlined the database management process and ensured consistency between the application's data model and database structure.

C# for Application Development

C# served as the primary programming language for developing the Blazor web app and other components of the project. Leveraging the latest features of C# 12 enabled the development of clean and efficient code to implement various functionalities.

Data Structures

Database Schema

The database schema for this project consists of three primary tables: “Readings”, “sql_sequence”, and “__EFMigrationsHistory”. The Readings table stores the attributes of the Readings collected from the energy meters. It stores the EnergyMeterId, MicrocontrollerId, the time at which the reading was captured, and the amount extracted from the reading. The sql_sequence table serves to track the last unique ID generated when a reading is added. Finally, the __EFMigrationsHistory table is a system table used by Entity Framework to track migration history for the database schema.

File System Structure

Given that the final version of this project is primarily a simulation, its file system structure includes a designated folder containing test images utilized for processing by the Python script. Beyond this, the organization of other folders within the file system is automatically managed by the dotnet framework, ensuring a streamlined and well-organized structure.

Interactions Between Data Structures

The Python script responsible for processing images and extracting meter readings generates data that aligns with the established data structure of the web application. However, it's worth noting that future modifications to the data structure may necessitate corresponding adjustments to the Python script, potentially introducing maintenance challenges. To mitigate this, a proposed enhancement involves integrating the script into the web application using C#, leveraging the framework's capabilities to achieve comparable functionality.

Conclusion

The Energy Insight Hub project has been a journey marked by challenges, learning opportunities, and valuable insights into the realm of software development and hardware integration. Over the course of the project, progress was made towards developing a comprehensive solution for remote energy meter reading and data visualization.

Despite encountering obstacles such as the inability to perform OCR on seven-segment digits and difficulties in hardware integration, important achievements were made. The successful extraction of meter readings from images using SSOCR, the development of a functional front-end interface with a live dashboard functionality, and the exploration of various software tools and frameworks demonstrate the project's potential for addressing real-world energy monitoring needs.

One of the most significant takeaways from this project is the importance of interdisciplinary knowledge. The challenges faced in hardware integration underscore the need for a deeper understanding of hardware components and their interactions with software systems.

Moreover, the iterative nature of software development highlighted the value of adaptability and continuous learning in overcoming obstacles and achieving project goals.

Looking ahead, there are opportunities for further refinement and expansion of the project. Future iterations could complete the integration of the hardware components and investigate advancements in OCR technology for improved digit recognition. In conclusion, the Energy Insight Hub project has been a rewarding endeavour that has deepened the understanding of energy monitoring systems, software development processes, and the importance of interdisciplinary knowledge.

Glossary

Energy Insight Hub: A software and hardware solution designed for remote energy meter reading, data extraction, and visualization, aimed at providing insights into energy consumption patterns.

OCR (Optical Character Recognition): A technology that enabled the conversion of different types of documents, including scanned paper documents, PDF files, and images into editable and searchable data.

Seven-Segment Display: A form of electronic display device for display decimal numerals. It consists of seven LED segments arranged in a rectangular fashion to form the numbers 0 through 9.

Linux Environment: An operating system environment based on the Linux kernel, characterized by its open-source nature, versatility, and widespread use in servers, embedded systems, and desktop computers.

Front-End Web Application: The user-facing part of a web application that users interact with directly in a web browser. It typically includes the user interface components and functionality for data presentation and interaction.

CLI (Command Line Interface): A text-based interface used to interact with computer programs by typing commands into a terminal.

Hardware Integration: The process of incorporating hardware components, such as sensors, actuators, or microcontrollers, into a software system to enable interaction with the physical world.

SignalR: A library for ASP.NET developers that simplifies the process of adding real-time web functionality to applications, allowing server-side code to push content to connected clients instantly.

Entity Framework: An object-relational mapping (ORM) framework for .NET applications that enables developers to work with relational data using domain-specific objects.

C# (C Sharp): A programming language developed by Microsoft within its .NET initiative, known for its simple syntax, type safety, and strong support for object-oriented and component-oriented programming concepts.

Blazor Web App: A framework for building interactive web UIs using C# instead of JavaScript. It allows developers to create web applications entirely in C# using Razor syntax.

Radzen Components: A set of pre-designed user interface components for Blazor applications, designed to simplify the process of building modern and responsive web applications.

SQLite: A lightweight, serverless, self-contained SQL database engine, commonly used in embedded systems, mobile devices, and small-scale applications.

FileSystemWatcher: A .NET class that enables applications to receive notifications about changes to the file system, such as when files or directories are created, modified, or deleted.

Tesseract OCR: An open-source OCR engine maintained by Google, capable of recognizing text in various image formats and converting it into machine-readable text.

Bibliography

Circuitdigest.com. (2019). *How to Use OV7670 Camera Module with Arduino Uno*. [online] Available at: https://circuitdigest.com/microcontroller-projects/how-to-use-ov7670-camera-module-with-arduino.

Auerswald, E. (n.d.). *Seven Segment Optical Character Recognition*. [online] Available at: https://www.unix-ag.uni-kl.de/~auerswal/ssocr/index.html.

n.a., (n.d.) [Online]. https://blacklionhvac.com/wp-content/uploads/electrical-meter.jpeg (Accessed November 2023).

Paredes, J. (2017). *Add WiFi to Arduino UNO*. [online] Instructables. Available at: https://www.instructables.com/Add-WiFi-to-Arduino-UNO/.

Harper, H.J. (2023). *Enable Ping Windows 10: How to Set Up Firewall to Allow It*. [online] Windows Report. Available at: https://windowsreport.com/enable-ping-windows-10/.

Rick-Anderson (2023). *Host ASP.NET Core on Linux with Nginx*. [online] learn.microsoft.com. Available at: https://learn.microsoft.com/en-us/aspnet/core/host-and-deploy/linux-nginx?view=aspnetcore-8.0&tabs=linux-ubuntu.

SulNR (2015). *Process.Kill() doesn't seem to kill the process*. [online] Stack Overflow. Available at: https://stackoverflow.com/questions/30249873/process-kill-doesnt-seem-to-kill-the-process.