Digital Twins for Microreactors

By

Sebastian Richards

A THESIS

Presented to the graduate faculty of

Computer Science

In Partial fulfilment of the Requirements for the Degree

MASTER OF SCIENCE

In

COMPUTING

2022

List of contents

List of Figures

List of Tables

Abstract

This dissertation entails the creation of a digital twin for a microreactor system by integrating a web based MERN stack application with node OPC UA to simulate the digital twin environment. Alongside the creation of the digital twin, the dissertation comprises of a summary as to what constitutes a digital twin as well the uses and benefits of the microreactor, with particular emphasis on the Vapourtec R-series microreactor.

The creation of this digital twin paves the way for future advancements in creating a type 2 digital twin for the Vapourtec R-series Microreactor used by the Chemistry department at Cardiff University, thus providing the benefits of remote use and data collection. The data collected can then be used to help better predict reaction outcomes in future experiments.

To achieve the creation of the digital twin an OOP model was adopted to encapsulate the individual components that comprise of a microreactor. These components can be customised by the user using the reactjs front-end. Using a combination of child process functions in the expressjs back end, the information regarding the components of the microreactor is written onto the OPC UA server which is subsequently initialised. Using the expressjs backend the application can then call a child process to call the node OPC UA client which interacts with the newly initialised OPC UA server demonstrating full integration of OPC UA within a MERN stack web application.

In summary the creation of the microreactor digital twin with OPC UA integration proved to be successful, establishing a solid foundation to be built upon in the future which in turn will allow for more accurate reaction predictions as well as remote use.

Introduction

Digital twins have existed since 2002, being first developed by NASA to better model physical-model simulation of spacecraft in 2010[1]. The purpose of a digital twin is to create a digital replica that encapsulates all physical aspects of something in the real world, e.g., a boiler, a warehouse, or as is the case in this project, a microreactor. This in turn allows for automation and data collection which can then be subsequently used for analysis. Another key aspect of the digital twin is that it is connected to the real-world appliance at all times, any operation of the real-world appliance will be known to the digital twin application, this is typically achieved via OPC UA (a machine-to-machine information protocol). Establishing this connection is perhaps the biggest obstacle when creating a digital twin due to potential incompatible software between the appliance, and digital twin client.

The Chemistry department at Cardiff university currently use microreactors with the R-series Vapourtec modular flow system to carry out chemical reactions on a small scale, for the duration of this project, these systems will be referred to as vapourtec R-series microreactors. They present numerous benefits over traditional batch chemical reactors which carry out the same reactions but on a larger scale all in a single reactor. Such benefits that the microreactor provides include the ability to keep the temperature consistent throughout the whole reaction column as opposed to localised 'hot spots' found in the batch reactors[2]. Other benefits include more homogenous mixing which leads to less product impurities[3]. Microreactors can be assembled in parallel on a large scale to match the output of a traditional batch reactor while still providing these benefits[4]. While microreactor Chemistry provides numerous benefits over traditional batch Chemistry, it is still a new technology that is yet to be fully understood that can produce unexpected Chemical products.

The desired outcome of this project is to create a digital twin that utilises full OPC UA integration with the long-term vision beyond the scope of this dissertation to connect the digital twin software to the vapourtec R-series Microreactor. The vapourtec R-series microreactor utilises OPC UA to enable machine-computer communication therefore it is vital that the digital twin software to be developed includes OPC UA integration. To link the digital twin being created to the vapourtec microreactor in the future it is worth noting that the current software that the R-series microreactors use in the Cardiff University laboratories (flow commander software) must be updated to the latest R-series software to allow the microreactors to interface with third party applications such as the digital twin software that is to be created. Shown in figure 1 is how the physical microreactor with the updated R-series software connects to OPC UA.

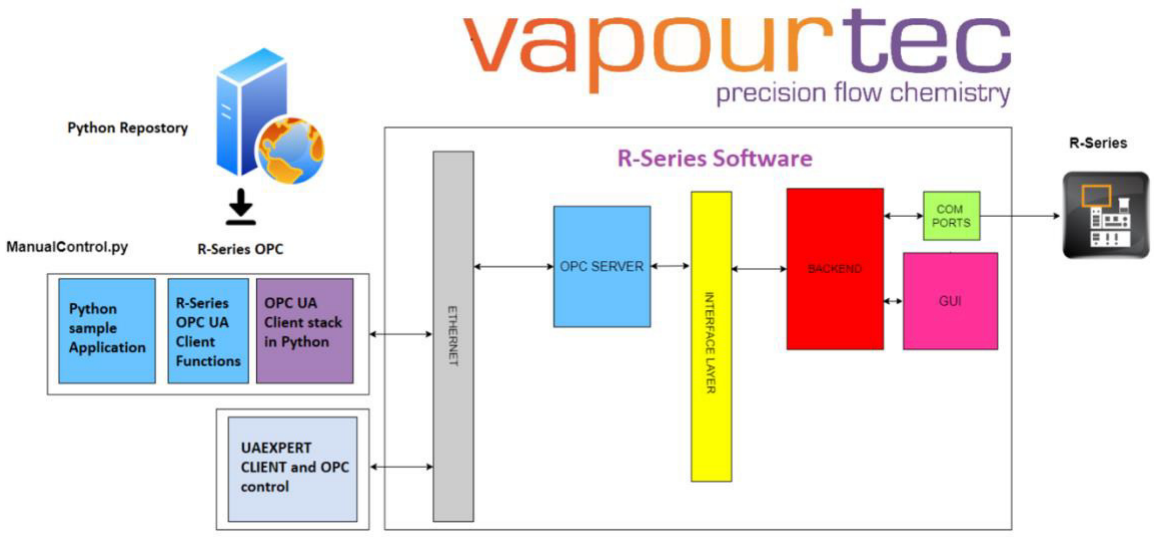

Figure 1: R series software architecture

By interfacing the physical microreactor with the digital twin software, real time data exchange can occur, sending the digital twin values corresponding to the current monitored reaction such as temperature, pressure and flowrate. With data collection, AI can be used to better explain the

reason for unexpected reaction outcomes that are often observed in microreactor Chemistry. Another benefit the digital twin software will provide is remote use, the digital twin client will be able to operate the microreactor remotely, an increasing need with remote work becoming more prevalent. A successful completion of this project would not only benefit the Cardiff Chemistry department but could potentially benefit and be used by other academic institutions globally that are investigating microreactor Chemistry. The list below shows what features are to be developed during the scope of this project regarding the digital twin software and what features will be integrated to the software outside the scope of this project, as discussed with the Chemistry department at Cardiff University.

To be developed in the scope of the project:

- Account authentication system
- OPC UA client-server integration and accessibility via the web application
- Create feature allowing the specification of a virtual microreactor that is to be instantiated and saved to OPC UA server
- Database integration that stores microreactor instances

To be developed outside the scope of this project:

- Integration of the physical microreactor to the web application
- Functions that allow remote operation of the microreactor via the web application
- Data collection between the physical microreactor and web application with the aim of future analysis

The digital twin application to be developed will be a web-based application using the MERN stack, a JavaScript stack comprising of mongoDB, expressJs, reactJs, and nodejs. A node software development kit, (node OPC UA) is used to create the OPC UA client and server. The frameworks chosen are to maintain homogeneity in the scripting language for ease of interpolation between the OPC UA client/server and web application. The web application will provide the user the ability to customise a bespoke microreactor tailored to that of a physical microreactor. From there a virtual replica of the microreactor can be created, and once created be saved to the OPC UA server, in turn initialising the server and realising a live OPC UA connection between the OPC UA server (now containing the bespoke specification of the desired physical microreactor) and the web client. This fulfils the requirement of establishing a digital twin, which can then be connected to the vapourtec microreactor once further additions are made to the software outside the scope of this project.

Thesis Structure

This thesis comprises the design, specification, implementation and testing of the digital twin software with the design & specification as well as the implementation sections utilising an SRS document. A background provides information as to what constitutes a digital twin, further detail regarding the microreactor system, OPC UA as well as an example of another digital twin system created for a pumping system while the future work section discusses further implementations that can be used to improve the software before interfacing with the microreactor system. As the literature regarding digital twins of microreactors is scarce, the thesis also provides a solid framework for the first step of designing this type of digital twin.

In summary, this thesis provides an in-depth overview on the development of a type 1 digital twin for a microreactor. Further additions to the software outside the scope of the project will allow for integration between the physical microreactor and the digital twin software allowing for remote use of the physical microreactor as well as data collection to be used in conjunction with AI algorithms to better prediction reaction outcomes.

Background

Recent investment in Cardiff university translates in the need for adaptation of current systems, with the microreactor system being one of them. This in time will create wider academic engagement transforming academic concepts into real world solutions. With such growth the need was identified by the Chemistry department to create an intelligent system that could extract, monitor and analyse data from their microreactor systems. It was established that a digital twin would be the best approach in solving this problem. The proposal arose from the need to create a more intuitive, economical and reliable way to carry out reactions via the microreactor system that would interact, compare and analyse reaction outcomes with the aid of accumulative data. Such facilitation will ensure a solid path for further research and competitiveness. At the time of the proposal there did not exist any digital twin for the microreactor, however, the first digital twin of the microreactor was recently created in late July 2022 by Idaho National Laboratory[5] but is still in early development. The lack of new material and information proved to be the constraints of this software implementation. At present, microreactor systems are limited and still in infancy although interest is rising, and new approaches are being developed.

This section takes a more in depth look into the individual components that the digital twin system will comprise of: The digital twin, OPC UA, Microreactors as well as an example of a digital twin system that was created for a similar pump system, respectively.

Background of the digital twin:

The digital twin market is currently valued at 6.9 billion USD as of 2022 and is expected to increase to 73.5 billion USD by 2027[6].

Currently there are 3 types of digital twin ranging from type 1 (supervisory) to type 3 (predictive). Shown below is a description of each type of digital twin.

Types of digital twin:

Type 1: Supervisory (or observational) digital twin.

The supervisory digital twin is the foundation upon which all digital twins are built upon. The criteria of this type of digital twin must be met for a piece of software to be considered a true digital twin and can later be further developed upon to create more advanced digital twins. In its purest form, a supervisory digital twin allows the passive flow of data from an appliance to the digital twin or vice versa which can then be monitored. The digital twin software consists of a virtual replica of the physical system which can be accessed through, typically a web application.

Type 2: Interactive digital twin

As previously mentioned, this builds upon a supervisory digital twin however an interactive digital twin also assumes some degree of control over the physical object.

Type 3: Predictive Digital twin

The predictive digital twin builds on the supervisory digital twin but will implement a high-level design using CAD or other software to help model the key features in a simulation when creating the digital twin. Alongside contextual analysis and in detail analysis, variables in the overall simulation can be changed to help see the simulations response. This in turn allows the digital twin to predict how a system or appliance would respond to different parameters being changed allowing businesses to make more financially sound decisions.

The digital twin type chosen for this project

In this project the digital twin being developed will be a type 1 (supervisory digital twin). A real time exchange of data will need to be established between the OPC UA server by accessing it from the OPC UA client via the web application. The purpose of this digital twin is to showcase the interoperability between the different software involved. This paves the way for further advancements outside the scope of this project where the type 1 digital twin will be upgraded to a type 2 digital twin, interfacing with the physical microreactor system and allowing remote operation as well as data exchange.

Benefits of the digital twin include:

1. Accelerated risk assessment and production time:

Digital twins can be designed to create a digital copy of an already existing appliance but can also be created before an appliance has been created to first test the feasibility before production takes place. Digital twins can be created to provide an opportunity for realistic training and to avoid any potential mistakes affecting the real-life appliance.

2. Predictive maintenance

An important aspect of the digital twin is to keep a real time exchange of data from the appliance to the digital twin. If sensors are in place to detect the long-term ware of a system, a digital twin will be able to detect these changes allowing proactive action to be taken to avoid damage to the system.

3. Real time remote monitoring

Digital twins can receive data in real time and this data can be stored in an accessible manner which can be assessed from anywhere. The accessibility of this data coupled with how it's stored makes real time remote monitoring another benefit of the digital twin.

4. Better financial decision making

Digital twins can monitor financial data such as the cost of materials and labour. Coupled with data analysis digital twins can help business owners make more financially sound decisions.

Background of the microreactor:

Microreactors are designed to carry out chemical reactions on a small scale, providing an alternative to traditional batch Chemistry with more control. The reagents are pumped into the reaction column via a tube where they subsequently react with the aid of the electrode plates (cathode and anode plates). Figure 2 shows the individual components that make up a microreactor.

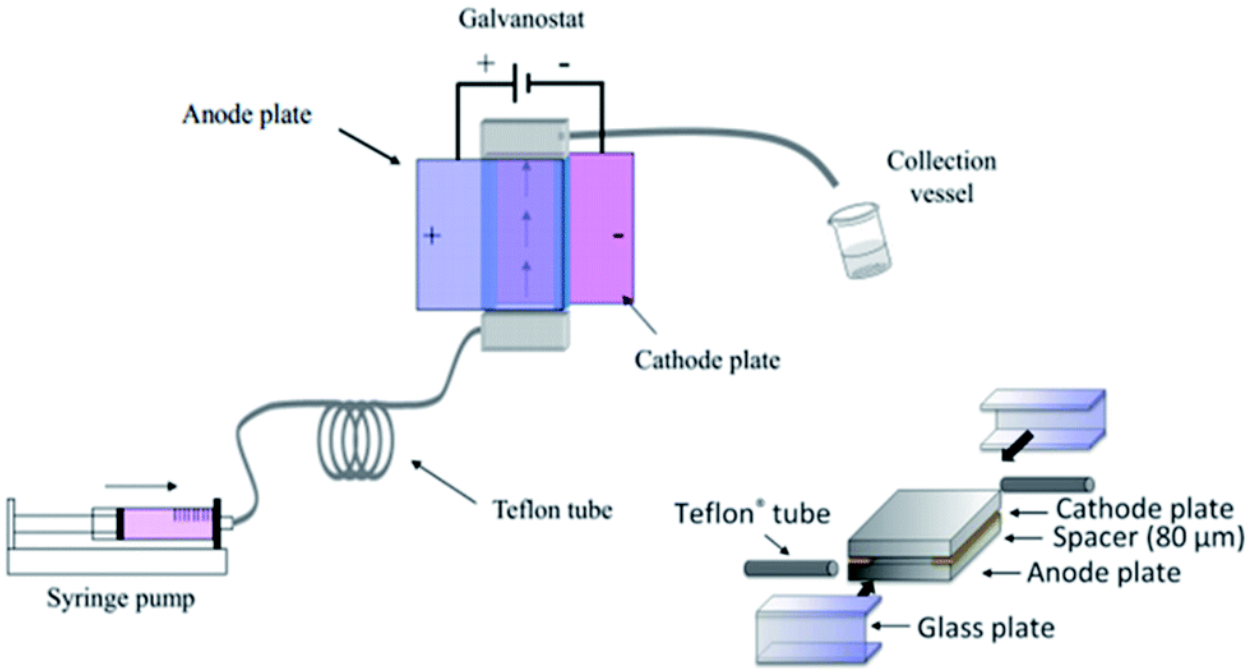

Figure 2: Microreactor components

While the diagram shows a syringe pump as the pumping system, the pumping system used at Cardiff university is the R-series modular flow system shown in figure 3. The R-series flow system provides a multitude of censors to monitor different variables such as temperature and flowrate

and it is through this system that the digital twin software will receive the respective data via OPC UA.

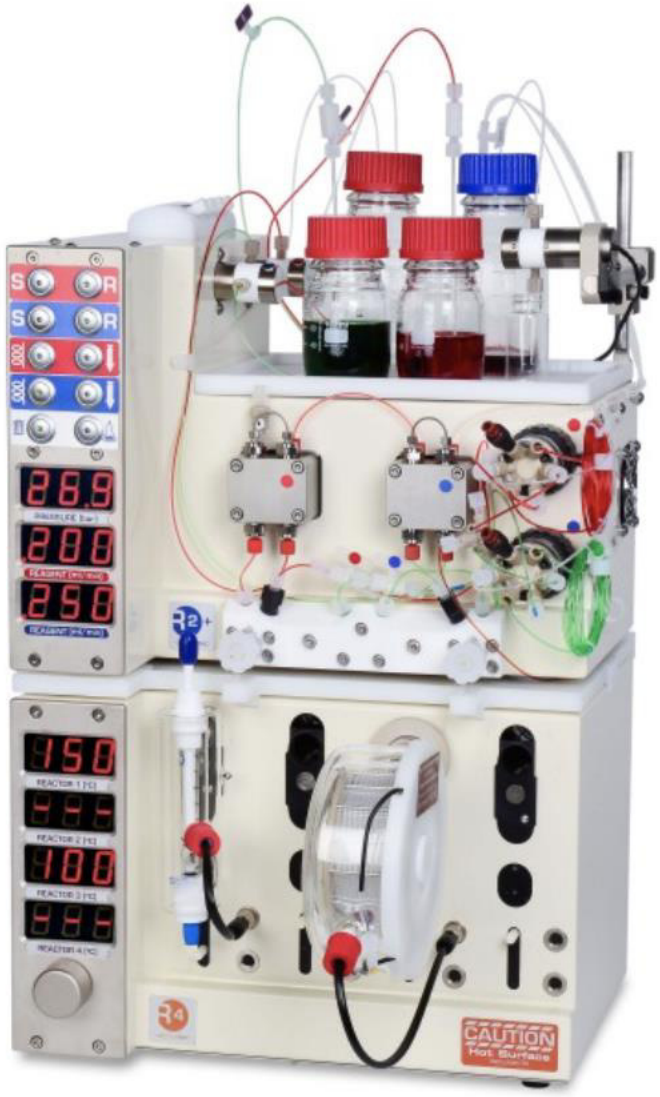

Figure 3: R-series flow pump

The R-series pumping system provides a continuous flow of reagents that pass through the electrodes to react, the rate at which it does so is called the flowrate. This is known as flow Chemistry and is an integral part of this specific type of microreactor system.

Figure 4 shows an example of a reaction that was carried out using the R-series microreactor at Cardiff University[7] and shows the electrodes radicalising the iodobenzene compound by removing a single electron from the iodine atom highlighted in blue making it more reactive and susceptible to establishing bonds with other reagents that it normally wouldn't do so, as is the case in this reaction with the weak nucleophile (compound that donates an electron pair to establish a new chemical bond) of OAc^-.

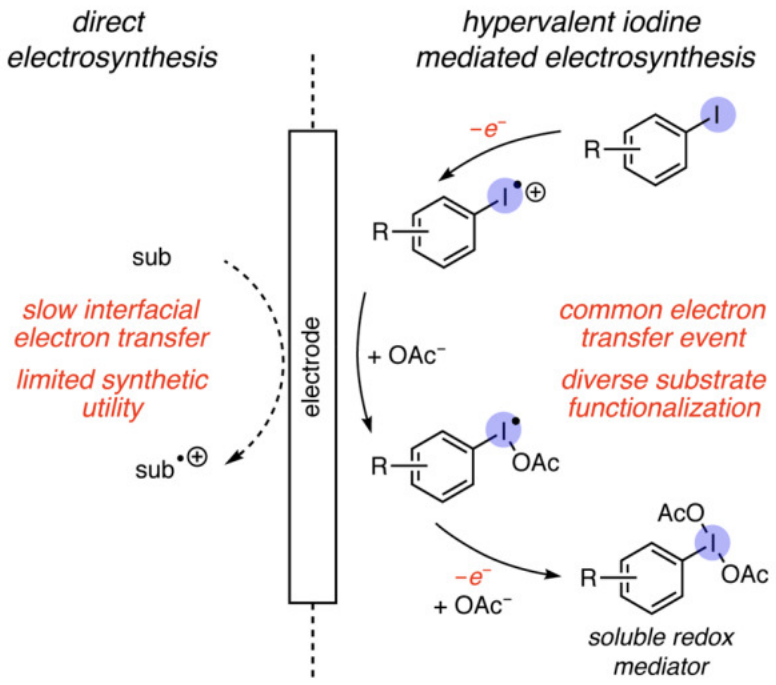

Figure 4: Electrosynthesis of hypervalent iodine

This is one example of many that microreactor Chemistry is capable of, particularly regarding flow Chemistry.

Flow Chemistry Microreactor Benefits:

Since microreactors allow for a continuous flow of reagents combined with the small surface to volume ratio, the cumulation of localised hotspots which are often a problem that traditional batch reactors are susceptible to is greatly reduced. Hotspots are regions within the reactor where more heat is produced due to a relative increase of localised reactivity, this can lead to undesirable side reactions or fragmentation[8]. The small surface to volume ratio of the microreactor allows for heat that is formed within the reaction to be cooled more efficiently via cooling, this greatly diminishes the formation of hot spots and allows for a more homogenous stable reaction.

Reactions can be carried out via the flow microreactor continuously via a continuous flow of reagents being pumped into the reaction chamber, this allows for the subsequent processing of unstable intermediates which in turn avoids work up delays.

Higher purity of product – Due to the smaller surface to volume ratio the microreactor observes a lower heat coefficient when compared to that of a traditional reactor. Depending on the temperature, chemical reagents can take different reaction pathways providing impurities. E.g for the chemical reaction between imidazole derivatives and sulfonates shown in figure 5 a purity of product is observed at a maximum of 96% observed in a batch reactor[9].

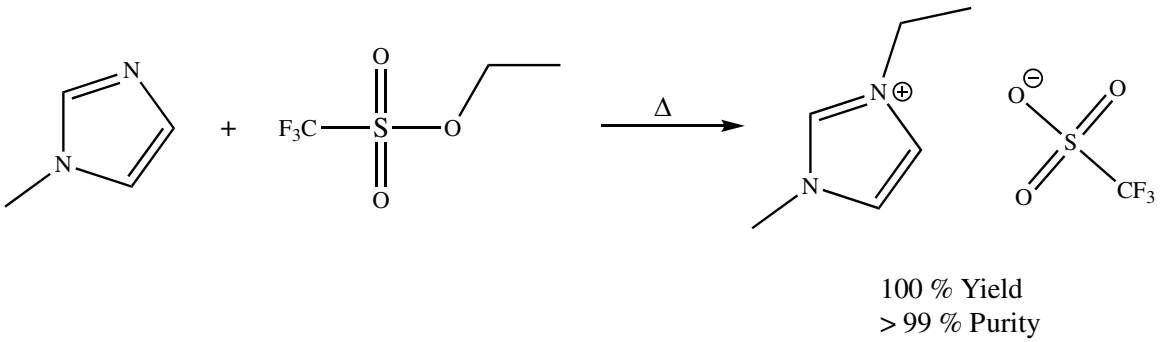

Figure 5: chemical reaction between imidazole derivatives and sulfonates

This is due to the highly exothermic nature of the reaction, which means heat is produced as a result of the reaction being carried out which stays in the reactor system for longer due to the higher heat coefficient present in batch reactors. The purity observed from the same reaction by using the flow microreactor system is observed at values exceeding 99%.

In summary, microreactors provide numerous benefits over traditional batch reactors such as higher product purity and better temperature control. Creating a digital twin to interface with the microreactor will allow for more advanced data collection and remote use, however, before this connection is to be realised a simulated environment must be established first in creating a type 1 digital twin which is what the aim that this project entails. The type 1 digital twin is to interface with OPC UA allowing for the upgrade to a type 2 digital twin to be more seamless with the Vapourtec flow system that is OPC UA compliant.

Background of OPC UA:

OPC UA stands for Open Platform Communication Unified Architecture and is a machine-to-machine communication protocol for industrial automation developed by the OPC Foundation.

Shown in figure 6 are the features and benefits that OPC UA provides.

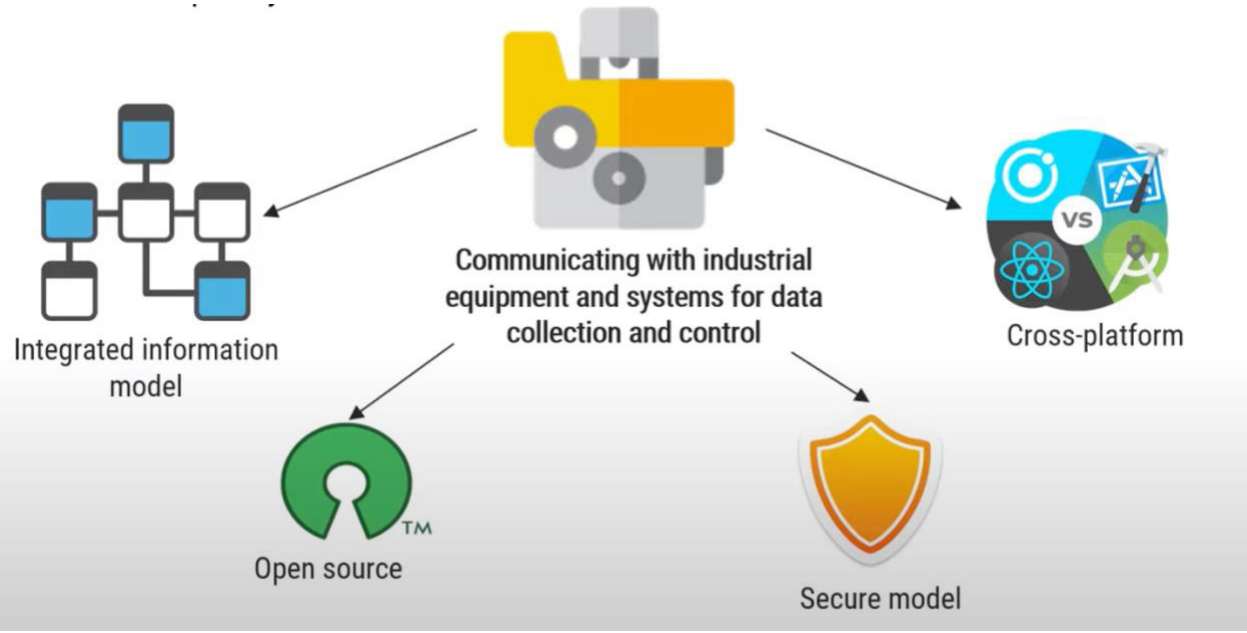

Figure 6: OPC UA features

Integrated information model:

The model of OPC UA has a standardised information structure that utilises a framework that can represent various types of data as objects in an address space, these objects consist of nodes connected by references and can subsequently be accessed via standard services. The primary objective of the OPC UA address space is to provide a standard way for servers to represent objects to clients. An example of this address space is shown in figure 25 demonstrating how information

is stored on the OPC UA server and shown in figure 20 is the client call to retrieve the respective information.

Open Source:

OPC UA is open source which has enabled many different pre-calibrated server-client models that can be downloaded. In this project the OPC UA server has been coded from scratch, but server simulations could provide an alternative during this stage of development for the digital twin software.

Secure Model:

OPC UA provides two layers of security via the communication layer and the application layer. The communication layer utilises encryption, signatures and digital certificates while establishing a connection with the OPC UA client. The application layer of security ensures the user is authenticated and verifies permissions.

Cross platform

OPC UA can be used on various operating systems allowing for cross platform server-client communication. This allows for scalability which makes it possible for future additions to be added to the project without needing to redesign the code (e.g., calibrating the physical microreactor to connect to the OPC UA server). OPC UA can be coded in many languages, most typically Java, however, many software development kits (sdks) exist which allows for OPC UA to be utilised using languages such as node JavaScript as is the case in this project through node OPC UA.

In summary OPC UA provides a secure way of establishing a connection and sending and receiving data via OPC UA client-server communication between different machines which is easily scalable.

Example of another digital twin for a similar system:

Before moving onto the planning and development phase for the software it is worth noting other implementations for similar digital twin systems. “Creating a digital twin for a pump”[10] is a short paper written by Chris Macdonald, Bernard Dion, and Mohammad Davoudabadi showcasing an example for a digital twin system developed to model a pumping system.

The system consists of a web application that acts as a digital copy of the pumping system. This web application is connected to the pumping system via an IoT (Internet of things) industrial platform. IoT describes the network of physical objects that are embedded with sensors, software, and other technologies for the purpose of the exhange and connection of data between other devices and systems connected to the internet. In the case of this project ThingWorx is used as the machine-to-machine communication protocol between the web application and IoT which connects to the physical appliance.

Through this system, data could be received from the physical pump and displayed on the web application establishing a real-time data exchange between the two entities. This connection allowed for remote use via any device type connected to the internet.

In the system to be developed a web application is to be created that is to connect to node OPC UA. It will be through OPC UA that the web application will be able to communicate with IoT and thus be able to connect to the physical microreactor system being able to receive data from its sensors (predominately those that are found on the Vapourtec R-series flow system). The aim of this project is to establish the first step in this process by creating a supervisory digital twin comprising of a web application which interfaces with an OPC UA server environment and will be done so by executing node OPC UA client calls via the web application.

Specification, Design & Implementation

This section covers the specification, design and implementation of the project. The section first looks at developing the digital twin software by specifying the requirements, proposing the designing and then carrying out the implementation of application.

To begin designing any software the desired functionality must first be considered. Listed below shows the functional requirements that the application must contain to fulfil the aim of the project:

Functional Requirements:

- User authentication system
- Web-based application
- Specify and create a virtual replica of the microreactor system
- View created microreactor objects (specific to each user)
- Integration with OPC UA server environment

To achieve realising these functional requirements a web application is to be developed which utilises a user interface that allows the user to specify a microreactor object. The web application should provide functionality for logging in and out as well providing the ability to specify a custom microreactor. A UML class diagram was created to decompose the microreactor system into smaller attributes. The attributes will be as follows as agreed upon during a meeting with the Chemistry department:

- Electrode 1, 2
- Reagent 1, 2
- Tubing (Tube length, Tube Diameter, Number of Loops, Material)
- Reaction conditions (Temperature, Pressure, Flow Rate)
- Variable Reaction Conditions (Reaction Conditions + Time Value)
- Electrode Dimensions (Electrode Distance, Electrode Area)

Shown in figure 7 is a UML class diagram which incorporates all the respective microreactor attributes.

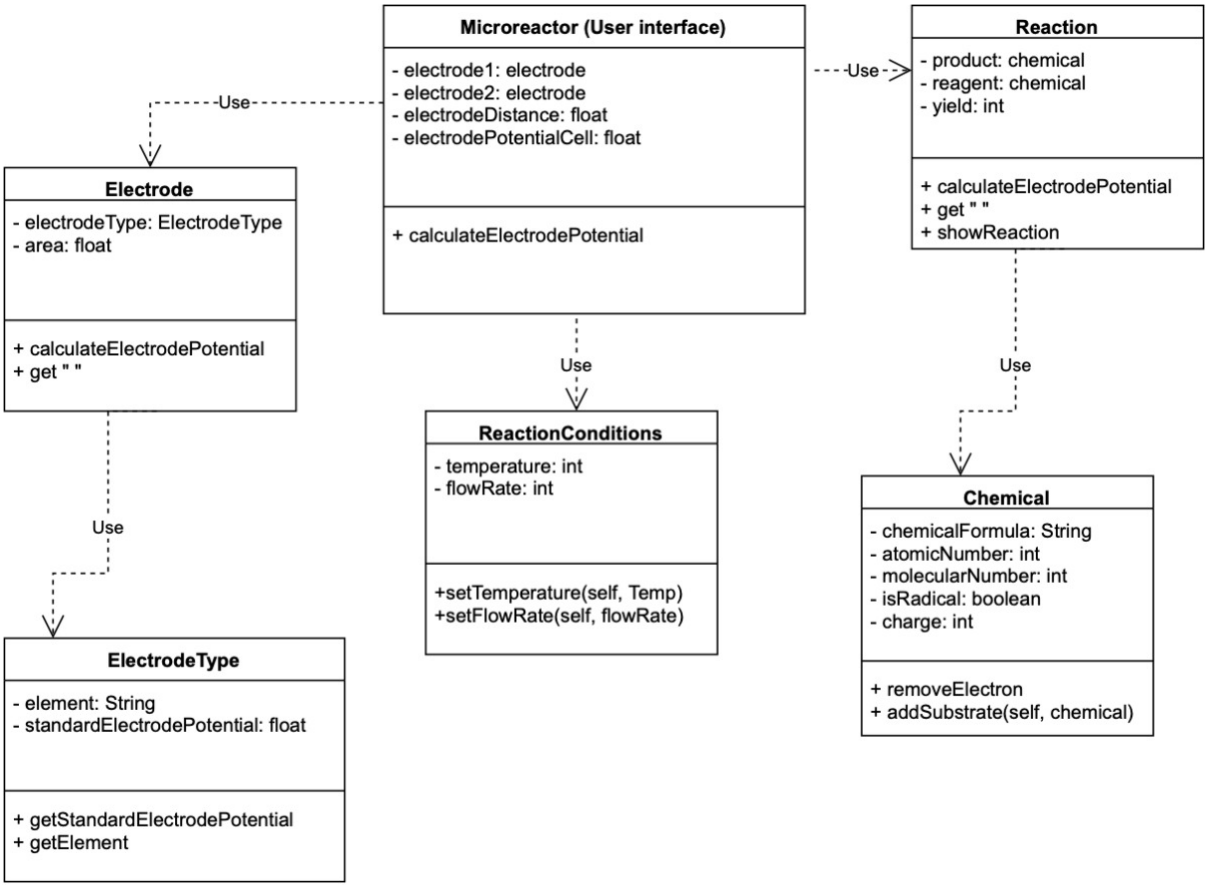

Figure 7: UML class diagram of microreactor object

Language choices: The MERN stack has been chosen to design the web application due to its scalability, relevance and homogeneity in scripting language. Node OPC UA will be used for both the OPC UA server with which the MERN stack application will interface with, and the OPC UA client which is what the MERN stack will use in order to communicate with the OPC UA server.

Shown in figure 8 is a high-level overview of how the application will interpolate between the various software components.

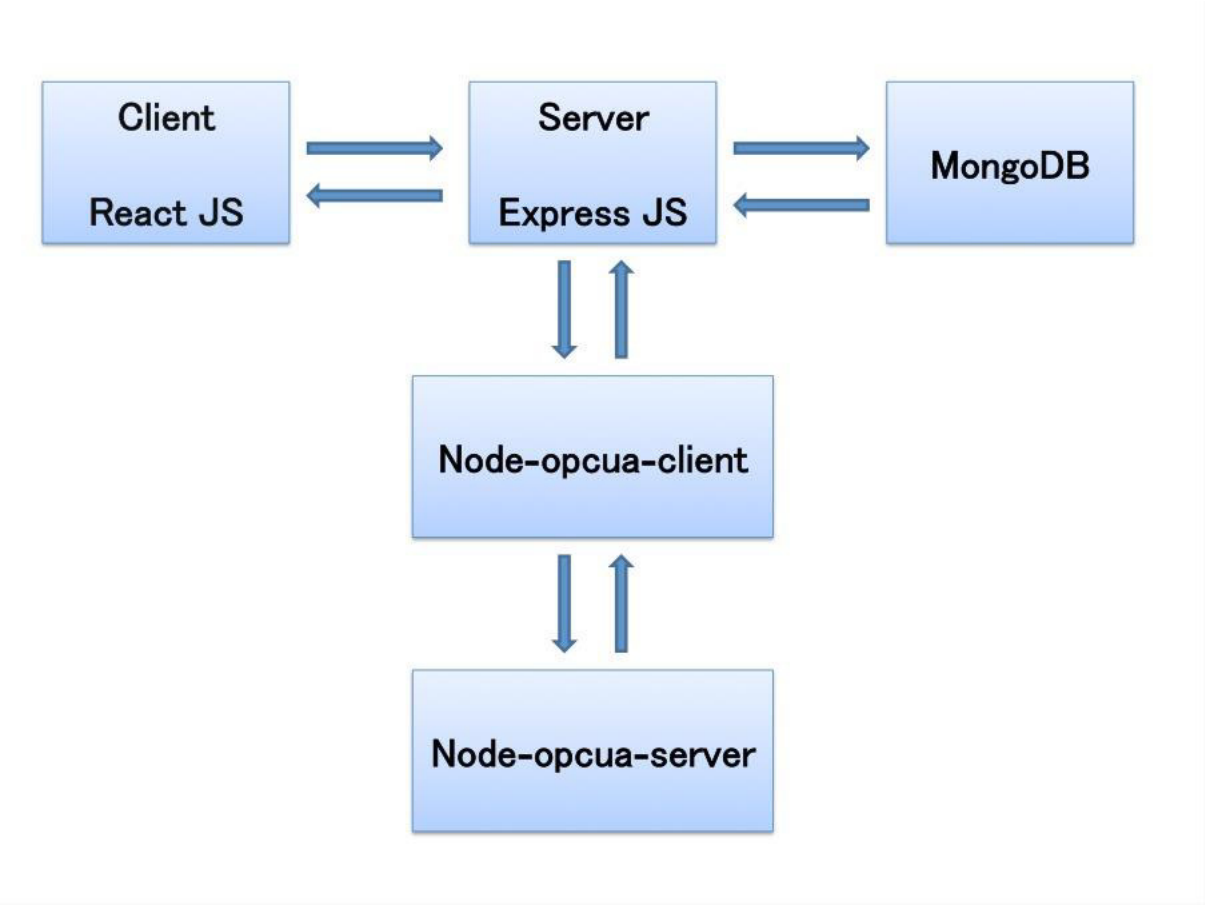

Figure 8: High level overview of application architecture

It is worth noting that the Node OPC UA server environment is to be initialised to run the server locally however, future additions to the software will connect the web application to a non-local OPC UA server.

By decomposing the microreactor into its constituent components, each attribute will be able to be specified by the user using the user interface, thus fulfilling the user requirement: Specify and create a virtual replica of the microreactor system.

The user interface must also include an authentication system which allows users to create an account and sign in providing the appropriate error messages. The main functionality of the application should demonstrate an interoperability between the MERN stack application and the node OPC UA server.

The design of the user interface is shown in figure 9, 10, 11 and 12. The following pages show the authentication system, the home page, the create page and the view page respectively. The implemented design covers the final product as well as the stimulus response sequences found from figure 14 onwards.

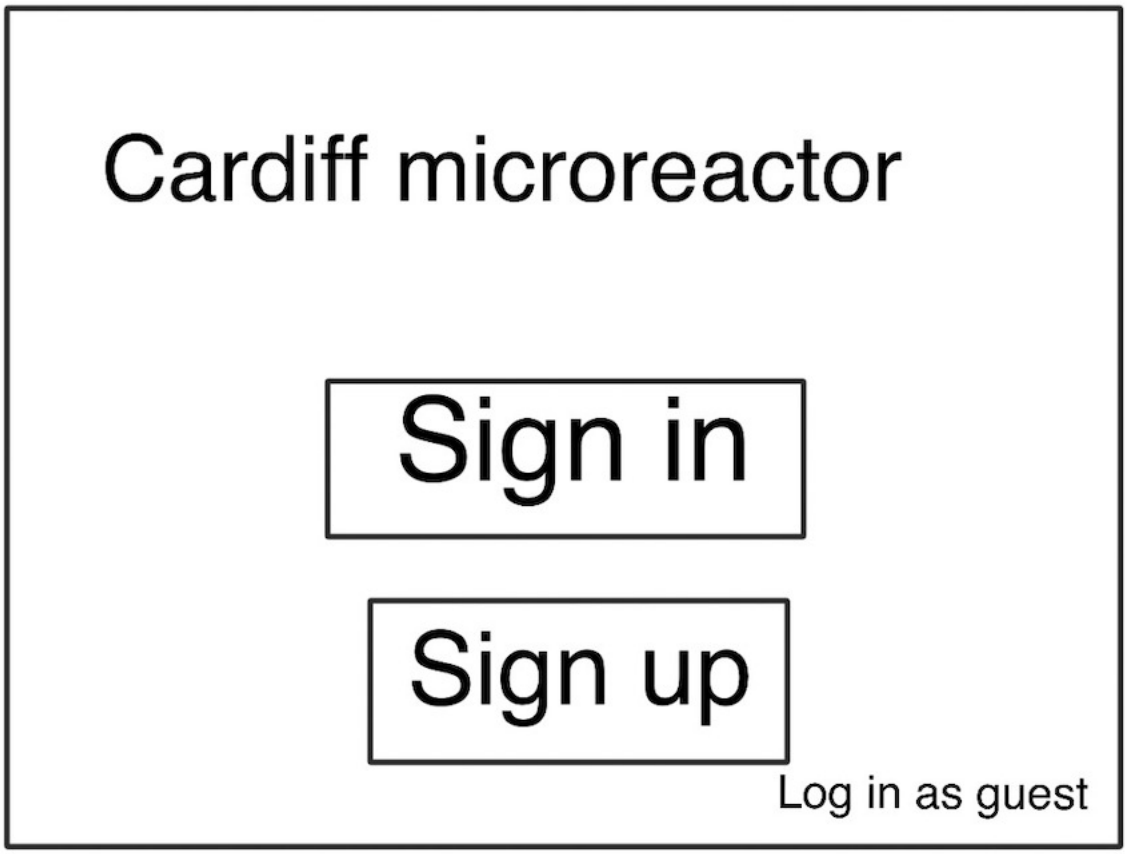

Figure 9: Design of authentication system

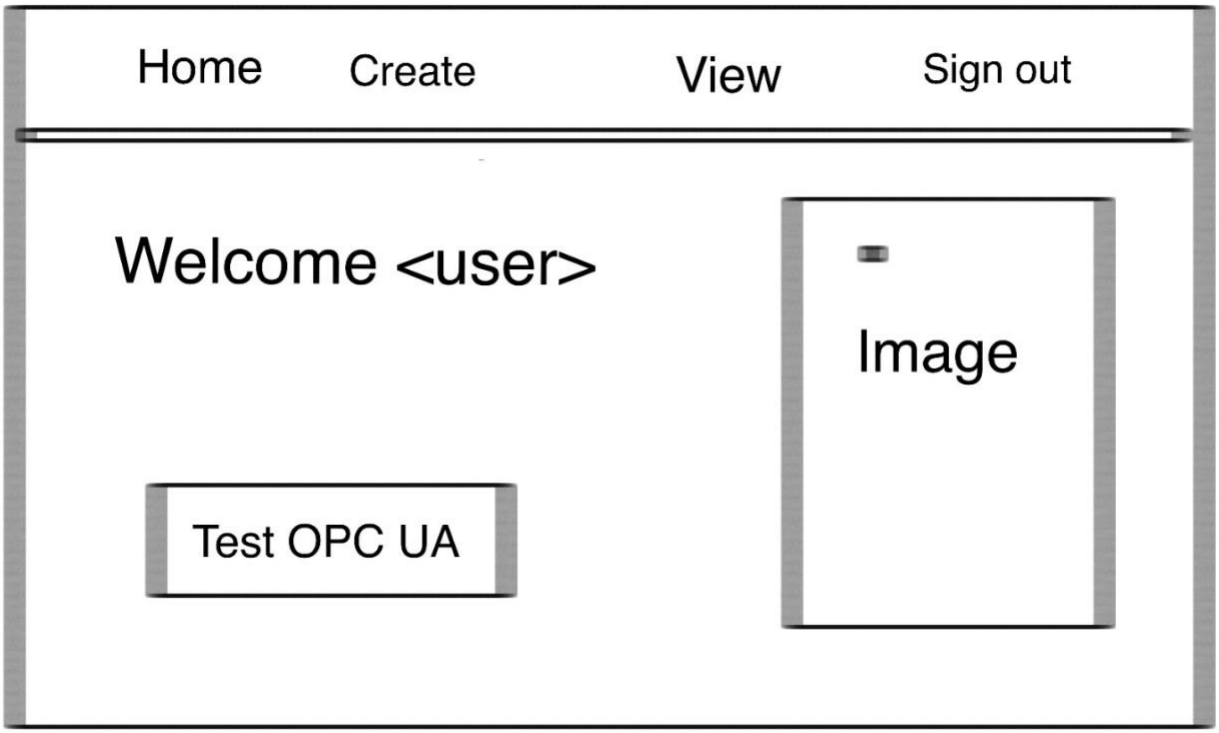

Figure 10: Design of homepage

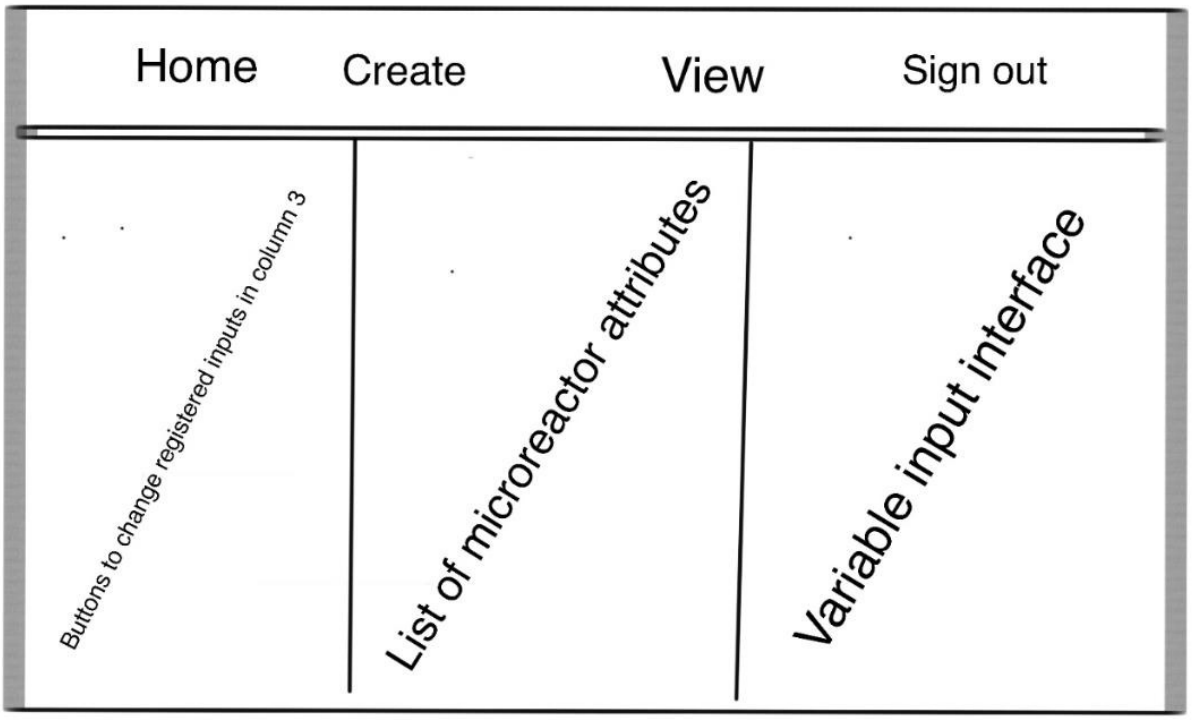

Figure 11: Design of Create page

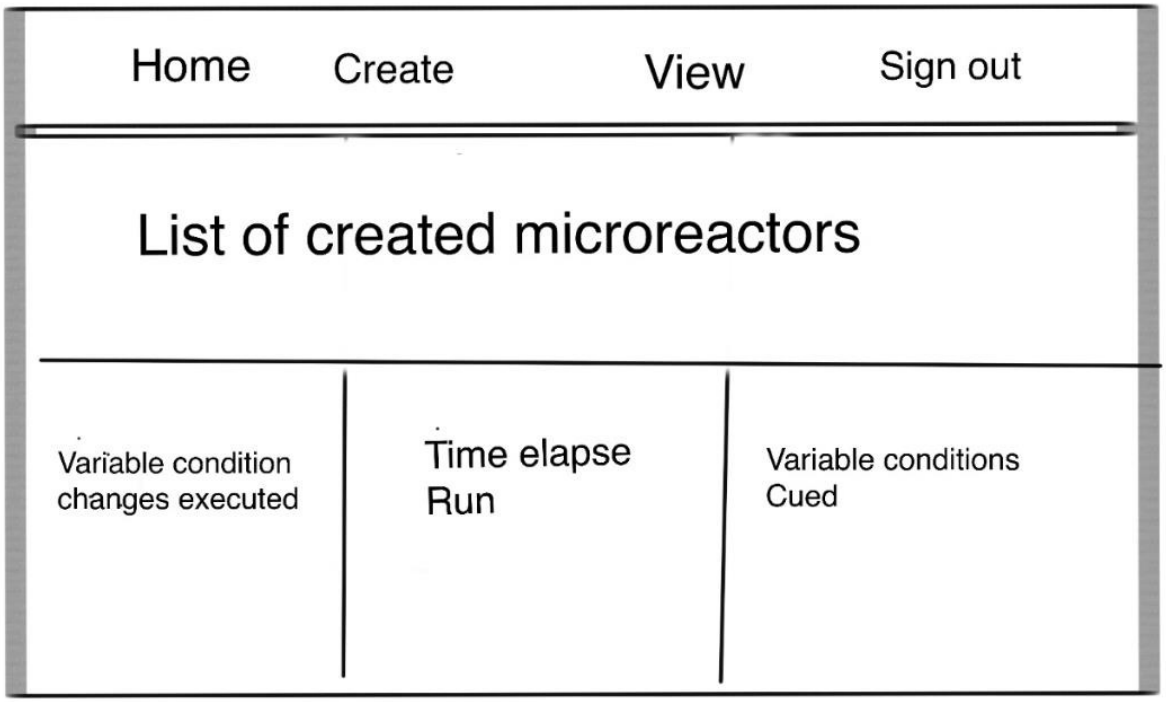

Figure 12: Design of view page

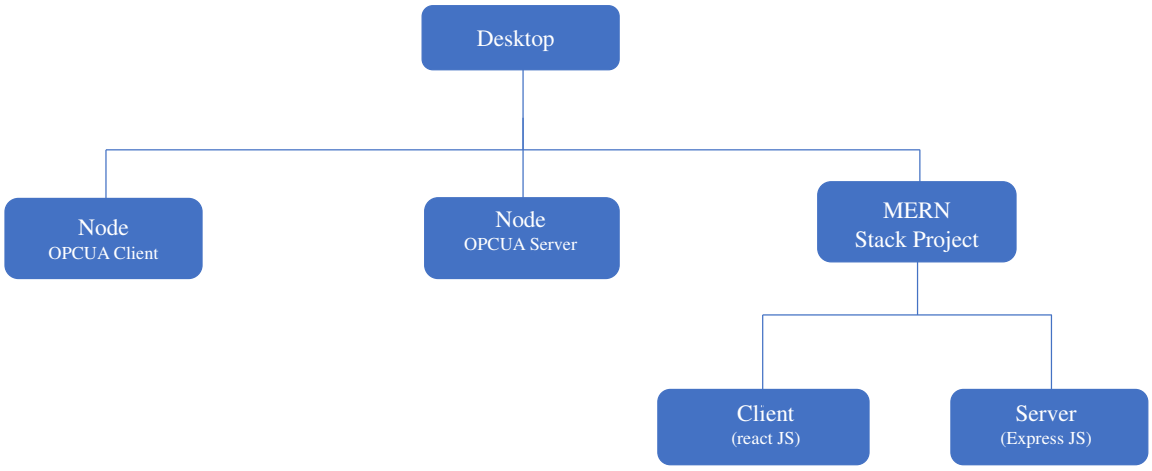

Figure 13: directory structure of application

The directory structure of the application will be as follows encompassing all components of the digital twin in the same directory for ease of interoperability.

Introduction

Purpose

The purpose of this SRS document is to describe the functionality of the application in detail with respect to the front-end web application to the back end expressJS functionality regarding OPC UA communication. The 1st edition of the software is contained within this document, which contains a fully functional MERN stack application which allows users to create a virtual replica of a microreactor, subsequently initiating the OPC UA server, allowing for OPC UA client-server connection via the MERN stack front-end reactjs GUI. This is an improvement on the initial virtual microreactor software that was coded in python with tkinter shown in the appendix.

Intended Audience and Reading Suggestions

The document is intended to be read by future developers, as well as users. The document contains a brief overview of the project scope and purpose, a more in-depth overview of the product with respect to requirements and constraints as well as product perspective and features. Users of the product can read the response/stimulus section of the document only as it contains an overview of the application functionality. The document also provides an overview of the different interfaces involved in the application, and how they interface with each other. For future developers it is recommended to the whole document to first grasp the capabilities of the application, a high-level overview of the application, and a more in depth look at how the application interpolates specifically in regard to OPC UA, respectively.

Project Scope

The scope of the application entails two main components, the MERN stack application and the OPC UA client-server side. The application allows the creation of multiple objects that contain the attributes of the virtual microreactors as well as reaction parameters that are

subsequently stored in mongoDB. After this is realised, an OPC UA server is started storing the temperature value of the microreactor created. An OPC UA client call can be made from the front-end of the application retrieving the newly stored information from the OPC UA server thus proving full OPC UA integration with the MERN stack application. This is a crucial first step which can then be further built upon in future additions to the software, however future additions will be outside the scope of the project in question. The key benefit that this software provides is establishing a foundation of a digital twin of a microreactor. With full OPC UA integration, the Vapourtec R-series microreactor can interface with the application once the R-series microreactor updates its software. Further additions to the software outside the scope of this project will entail the realization of this integration alongside more advanced data collection and security advancements before AI modelling to better predict reaction outcomes.

Product Perspective

The software being designed is a self-contained product and the first in its development line. Initially a virtual microreactor was developed using tkinter (see figure 40 in the appendix) however, due to lack of OPC UA functionality and the inability of tkinter to be deployed to the internet the project was redesigned using the MERN stack

Product Features

The major features that the product contains is the customisation of a virtual microreactor, the initialisation of the OPC UA sever, and performing an OPC UA client function call via the MERN stack front-end interface. These features are explained more thoroughly in the response sequence. Shown in figure 38 shows a context diagram and data flow diagram respectively giving a more in-depth insight into how these operations function with respect to the dataflow of the whole system.

User Classes and Characteristics

At the current state of development all users have the same privilege in specifying the attributes of the microreactor instance as well as initiation of the OPC UA server.

Operating Environment

The software can be used via any computer that runs on Linux or OS systems but not windows. This is due to the nature of the shell commands that are executed during the create microreactor and test OPC UA connection shown in figure 19.

Design and Implementation Constraints

Constraints that apply to the software require that users must be using a system that operates on either Linux or mac OS operating systems due to the scripting of the shell command files calling the command node to initiate the server and client files. Another constraint lies in the number of databases that can be created as microreactors are stored in mongoDB Atlas, a cloud-based database with a maximum capacity of 5GB. Fortunately, mongoDB can be updated and scaled to allow for more storage without recalibration of the database.

Assumptions and Dependencies

It is assumed that out the scope of this project that the physical microreactor communicates to a computer via OPC UA. It is assumed that the microreactor can integrate with the node sdk for OPC UA in node OPC UA to enable this connection and interoperability.

Dependencies

As mentioned in design and implementation constraints, it is vital that computers operate on Linux or mac OS operating systems.

System Features

This section illustrates and explains the key services provided by this application. The order starts with authentication and then the private pages, home, create and view respectively outlining the main functionalities that each route contains.

Sign In/ Sign Up Feature

1.1 Description and Priority

Upon entering the web application, figure 14 is presented to the user, allowing the user to either sign up, sign in or login automatically as the pre-installed user “guest”. The user authentication system works by using react useState in conjunction with local storage. A key-value system is stored in the local storage which the useState object retrieves. Any newly created users via signing up will be added to the local storage. If the user enters an existing key-pair value upon signing in, the user will be authenticated and signed in as that specific user, granting them access to the application.

Figure 14: Final authentication system design implementation

1.2 Stimulus/Response Sequences

Stimulus: Clicking the “Sign Up” button.

Response: Login interface renders sign up component shown in figure 15.

Stimulus: Clicking the “Sign In” button.

Response: Login interface renders the sign in component shown in figure 16.

Stimulus: Clicking the “Login as guest”

Response: User is authenticated as guest, and is redirect to the home page shown in figure 17 via react-router-dom.

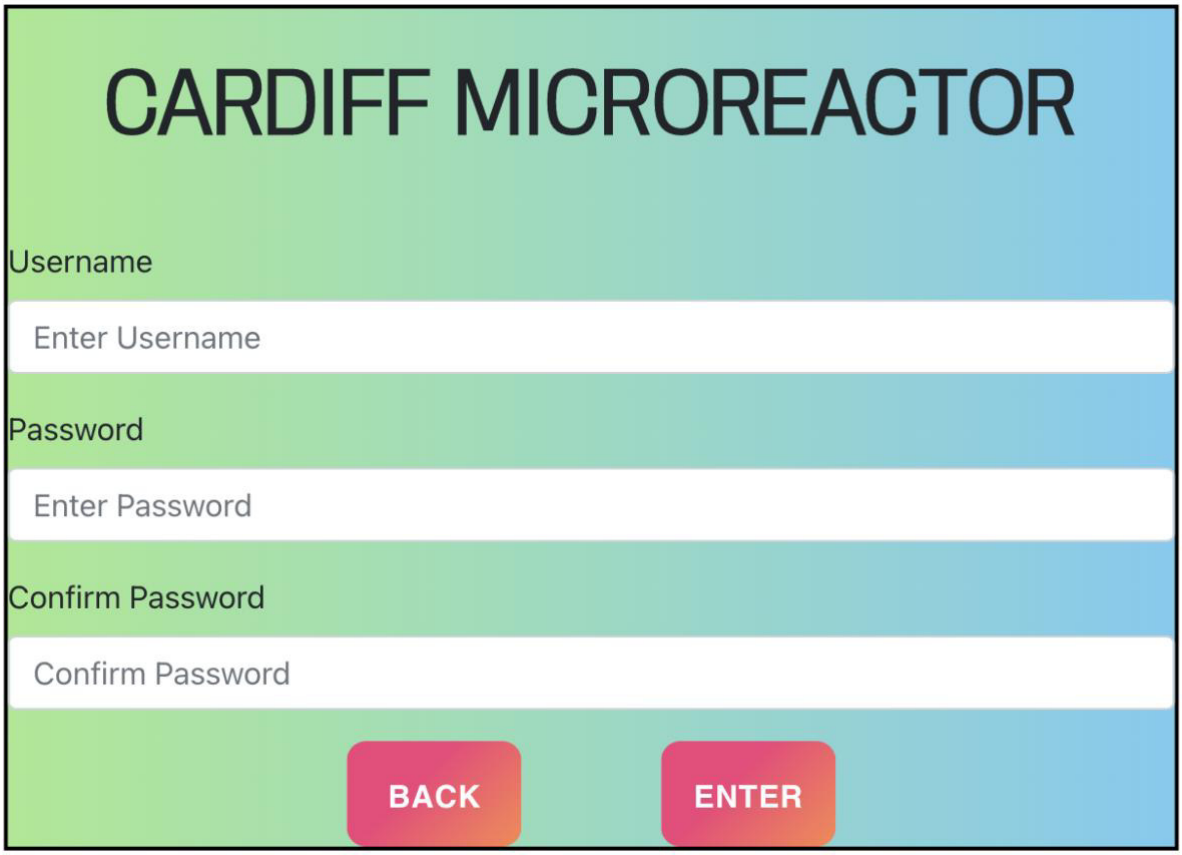

Figure 15: Authentication system after pressing “Sign Up”

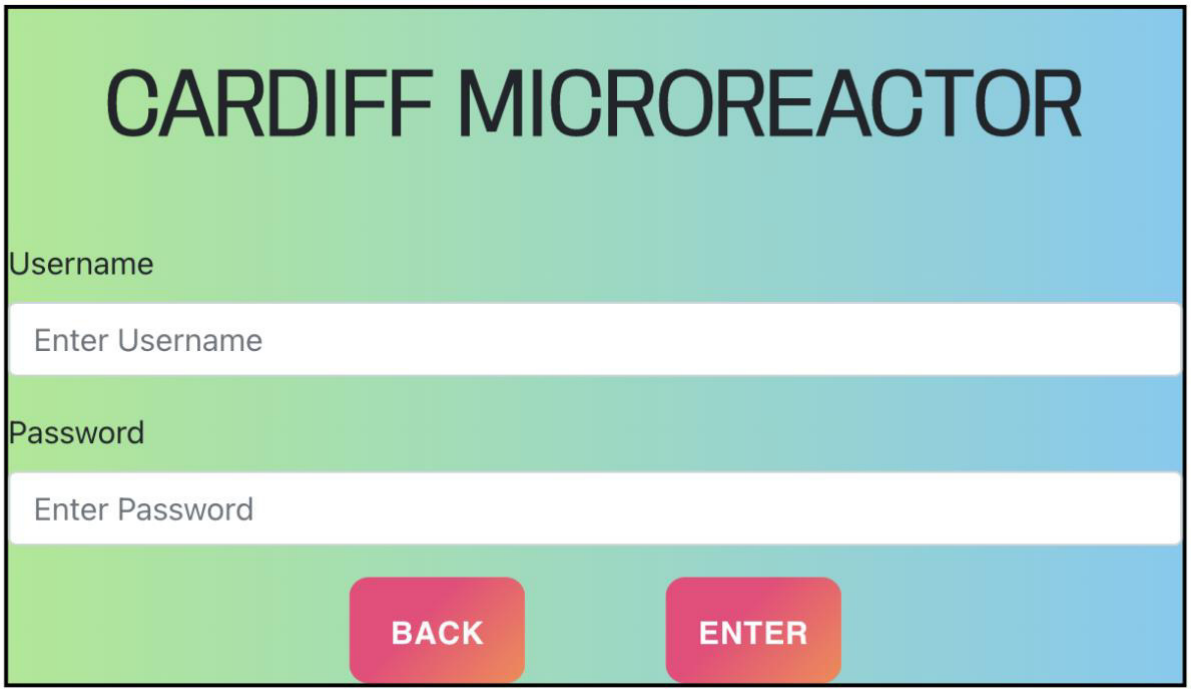

Figure 16: Authentication system after pressing "Sign In"

1.3 Functional Requirements

The appropriate requirements need to be satisfied for the user to successfully create an account using the sign-up component shown in figure 15.

Req 1: User must enter a username that doesn't already exist in the local storage. If the requirement isn't met an error is displayed to the user stating, "Username already taken".

Req 2: User must enter matching passwords for both fields: "Enter password" and "Confirm Password". If the requirement isn't met the error "Passwords don't match" is displayed to the user.

If both requirements are satisfied then the new user is successfully created, all previous error messages are cleared, and the user is re-directed to the original authentication component shown in figure 14.

The appropriate requirements need to be satisfied for the user to successfully sign in using the sign-in component shown in figure 16.

Req 3: Username must already exist in the local storage, else the error “Username doesn’t exist” is rendered and displayed to the user.

Req 4: The username and password must be an existing key-value pair in the local storage, else the error “Details do not match” is rendered and displayed to the user.

Home page/Test OPC UA connection feature

2.1 Description and Priority

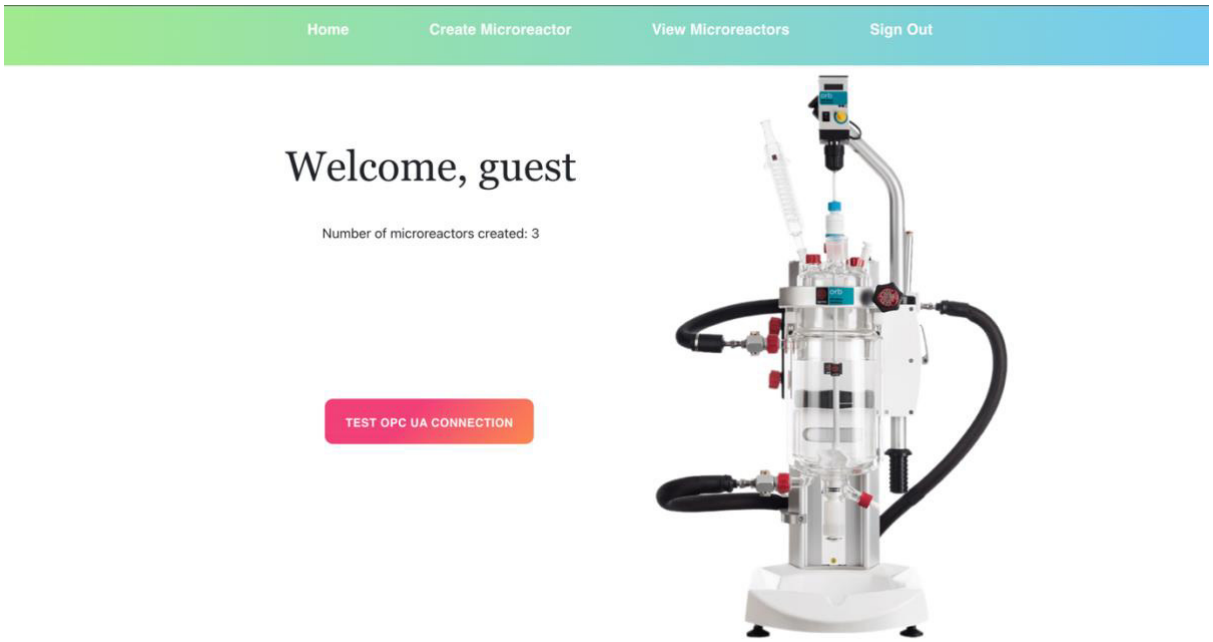

Figure 17: Home screen

Once authenticated the user is redirected to the home page shown in figure 17. The user is welcomed with the message “Welcome <username>” and the user can see how many microreactor instances have already been created on their account. The user can access the

“create microreactor” page, the “view microreactors” page and sign out and terminate their session by clicking sign out. Most importantly, the user can test to see if an OPC UA connection has been established within the application by clicking the “TEST OPC UA CONNECTION” button which alerts the user the status of the connection by testing the response from an OPC UA client call triggered from back-end API function call in expressJS “/testOpcua” shown in figure 18.

```
app.get("/testOpcua", (req, res) => {




    exec('sh ../../sample_client_ts/initiate.sh', (error, stdout, stderr) => {
        if(error) {
            console.log(stderr);
            res.send(`Not connected error: ${error}`);
            throw error;
        }

        if(stdout.substring(0,8) === "An error") {
            res.send("Error, please check terminal for details");
            console.log(stdout)
        } else {
            res.send(`Connection successful.`); }
            console.log(stdout)
        return;
    })


});
```

Figure 18: testOpcua back-end call

The function then uses a child process to execute the shell file located in the directory on the OPC UA client. The shell file to be executed is shown in figure 19.

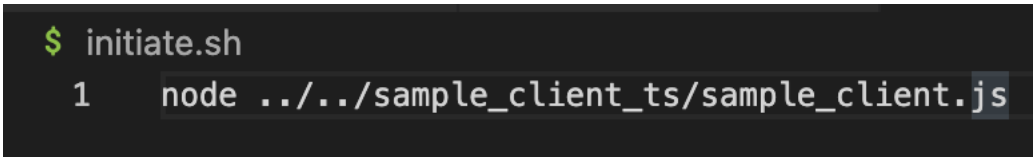

Figure 19: Shell command triggered after 'testOpcua'

The file is executed in the expressJS back-end and runs the OPC UA client file located in the sample_client_ts directory. The OPC UA client file is shown in figure 20.

```
const dataValue2 = await session.read({
    nodeId: "ns=1;s=temperature",
    attributeId: AttributeIds.Value});
console.log(" value = " , dataValue2.toString());
```

Figure 20: node OPC UA client call to read temperature value found in OPC UA server

The function is called and reads the nodeID which correlates to the temperature of the saved microreactor (if one has been saved). To successfully carry out the function the OPC UA server must be initiated and connected which is done so by creating a microreactor. The purpose of this function is to determine whether the OPC UA connection has been established or not. If the function is unable to read the variable, an error is produced and displayed to the user via the web application. If the variable is read, then an OPC UA client-server connection has been established and is alerted that the connection was successful, as well as showing the temperature value of the selected microreactor in the terminal.

2.2 Stimulus/Response Sequences

Stimulus: clicking on "TEST OPC UA Connection button"

Response: The user is displayed a message stating whether the connection was successful or not. A successful connection displays the message “OPC UA connection successful” whereas an unsuccessful message displays the message: “error, please check terminal”. The terminal then displays more information regarding the error.

2.3 Functional Requirements

Req 1: The user must have the OPC UA client and server files located in the same repository that contains the MERN application in order to successfully execute the child processes of the shell commands shown in figure 13.

Create Microreactor Function

3.1 Description and Priority

The create microreactor function is carried out by clicking the save button displayed in the bottom right corner of figure 14 and allows the user to create a microreactor instance defined in the mongoDB schema which is stored in mongoDB. The user must first fill out all the respective fields before activating this function. By filling out the respective attributes of the microreactor instance using the GUI a react state is updated containing all the respective information. The react state is then passed through to the backend and saved as a microreactor object in mongoDB. Saving the microreactor more also initiates an OPC UA server session which is automatically connected to the OPC UA client.

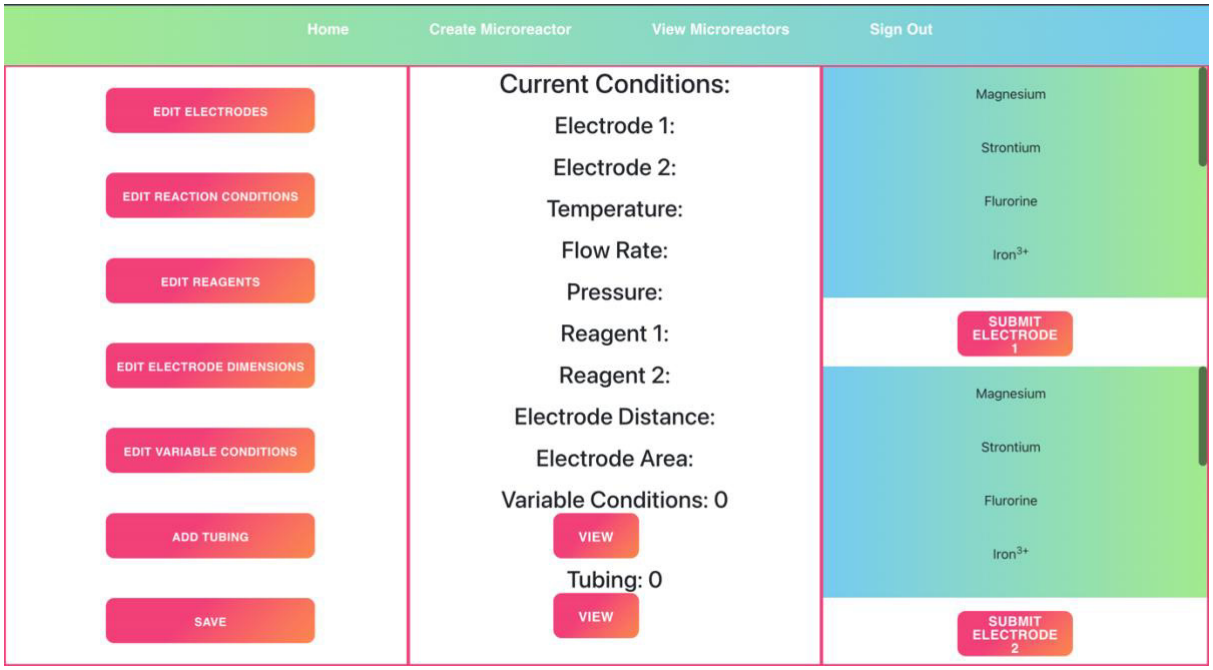

Figure 21: Create page

3.2 Stimulus/Response Sequences

Stimulus: Save Button is clicked

Response: back-end function "/postMicroreactor" is called shown in figure 22, saving the microreactor object to the mongoDB. The function writeOpcuaServerT is then called editing the OPC UA server file as shown in figure 23. The temperature attribute of the microreactor is added to the server file shown in figure 24 and then the OPC UA server file is initiated by calling the runOpcuaServer file shown in figure 25. A child process is executed executing the initiate.sh file which contains instructions to start the OPC UA server as shown in figure 26.

```
app.post("/postMicroreactor", async (req, res) => {
    const mr = req.body;
    const newMr = new MicroreactorsModel(mr);

    //updating the server file
    writeOpcuaServerT(mr.temperature);
    await newMr.save();
    //running the server file through child process
    runOpcuaServer()
    res.json(mr);



});
```

Figure 22: Overwriting then running the OPC UA server file

```
const writeOpcuaServerT = (temp) => {

    fs.readFile('../../myserver/sample_server.js', 'utf-8', function (error, contents) {
        if (error) {
            console.log(err);
            return;
        }


    const replacedT = contents.replace(/let temperature =/g, `let temperature =${temp};//`);



    fs.writeFile('../../myserver/sample_server.js', replacedT, 'utf-8', function (err) {
        if (err) {
            console.log(err);
        }

        });



    });

}
```

Figure 23: Overwriting OPC UA function

```
let temperature =20;//13;

// temperature
namespace.addVariable({
    componentOf: microReactor,
    nodeId: "ns=1;s=temperature",
    browseName: "temperature",
    dataType: "Double",
    value: {
        get: function () {
            return new opcua.Variant({dataType: opcua.DataType.Double, value: temperature });
        }

    }
})
```

Figure 24: Overwriting the temperature value in the OPC UA server file

```
//run the server
runOpcuaServer = () => {
    exec('sh ../../myServer/initiate.sh', (error, stdout, stderr) => {
        if(error) {
            console.log(stderr);
            throw error;
        }
        console.log('stdout', stdout)
    })
}
```

Figure 25: Running the OPC UA server

```
node ../../myserver/sample_server.js
```

Figure 26: Shell command triggered through runOpcuaServer function

Stimulus: Clicking any column 1 button.

Response: The following components are rendered in column 3

Stimulus: Clicking column 2 buttons

Response: The follow components are rendered in column 2

3.3 Functional Requirements

Req 1: For the save function to be released the electrodes, reaction conditions, reagents, electrode dimensions fields must all be submitted. If the condition is not met, an error will be displayed to the user alerting them of the respective microreactor field that needs to be submitted.

Req 2: The user must have the OPC UA server file in the correct location in their operating system as the writeOpcuaServerT function reads and writes the file in a fixed directory.

View/Delete/Run Feature

4.1 Description and Priority

The view, delete and run features are accessible via the view page shown in figure 27. Any microreactors that have been created by the user will be displayed to them by pulling the data from mongoDB. Every microreactor has the option to either be viewed or deleted. Viewing the microreactor gives the user information regarding the attributes of the microreactor instance as well as the staged variable conditions. The user can also delete the microreactors by clicking on the delete button, this also removes the respective microreactor instance from the mongoDB database. The run button when clicked starts a timer rendered under “time elapsed”, this is to be further developed once the physical microreactor is connected to the OPC UA server.

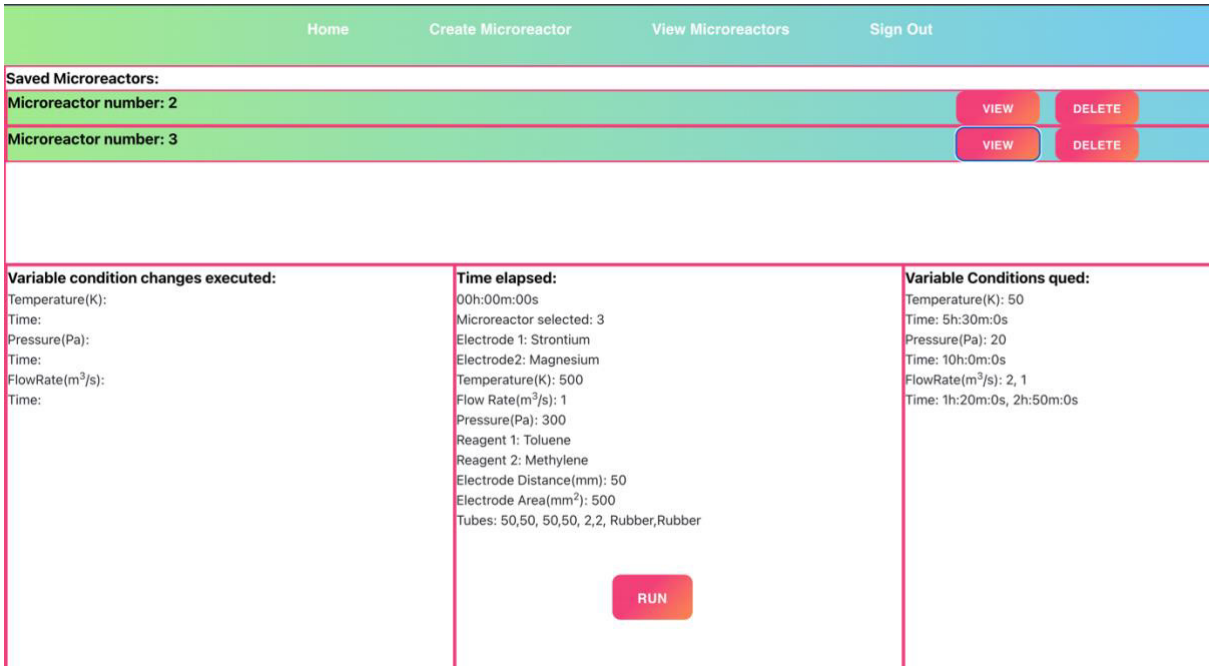

Figure 27: View page

4.2 Stimulus/Response Sequences

Stimulus: View button is pressed

Response: Bottom middle column and bottom right column are populated with the appropriate microreactor instance data accordingly

Stimulus: Delete button is pressed

Response: The respective microreactor is deleted from the mongoDB database

4.3 Functional Requirements

Req1: In order to carry any of the run, edit delete functions a microreactor object must first be instantiated using the create a microreactor. Clicking on the run button without a microreactor being selected will generate an error displayed to the user stating, “Please select microreactor”.

External Interface Requirements

User Interfaces

This section further discusses the further functionality of the user interface. The user interface consists of a private page (accessible once authenticated) and a public page (the authentication screen). The private page consists of multiple routes in Home, Create and View all containing a navbar fixed at the top of the page to allow for ease of navigation throughout the application. The response sequence pattern of the navigation bar is as follows:

Navbar functionality:

Stimulus: clicking on “Create Microreactor”

Response: Redirect to “Create” page.

Stimulus: clicking on “View Microreactor”

Response” Redirected to “View Microreactor” page.

Stimulus: clicking on “Sign Out”

Response: Authentication session is terminated, and the user is redirected to the authentication display shown in figure 14.

The home, create, and view routes are shown in figures 17, 21 and 27 respectively. Since the functionality of the home and view functions have already been summarized in the response/stimulus section the rest of this section will only further discuss the additional functionality of the create route.

The create page is split into three columns, column 1 displayed on the left which provides buttons to be clicked which render the correlating component in column 3. E.g., clicking on "Edit Reaction Conditions' will render the component displayed in figure 34 in the third column as opposed to the edit electrode component rendered by default. Column 2 updates when information has been submitted in column 3 using useState in reactJS. Since there is an unknown amount of tubing and variable conditions that correspond to the microreactor instance, displaying the information in column 2 could be potentially troublesome if the user inputted dozens of different variable reaction conditions. The user can click on the view buttons displayed under "Variable Conditions" or "Tubing" which will replace column 2 with the columns shown in figure 29 and figure 28 respectively to avoid clogging the default column with all the displayed data.

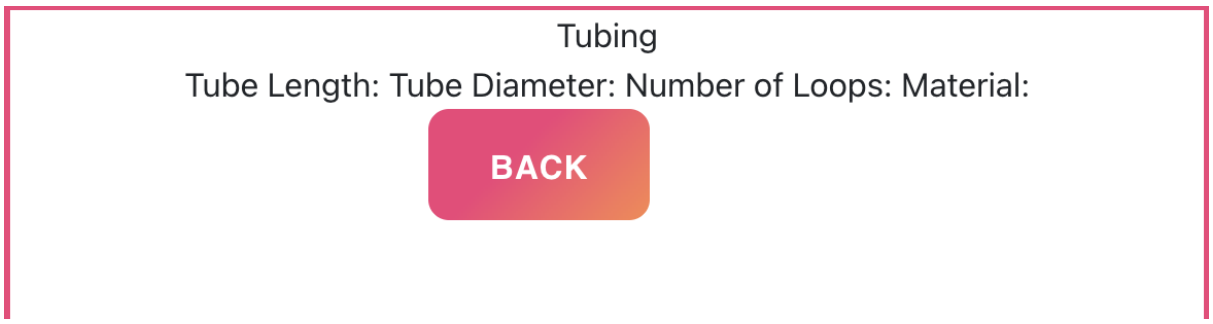

Figure 28: Tubing interface

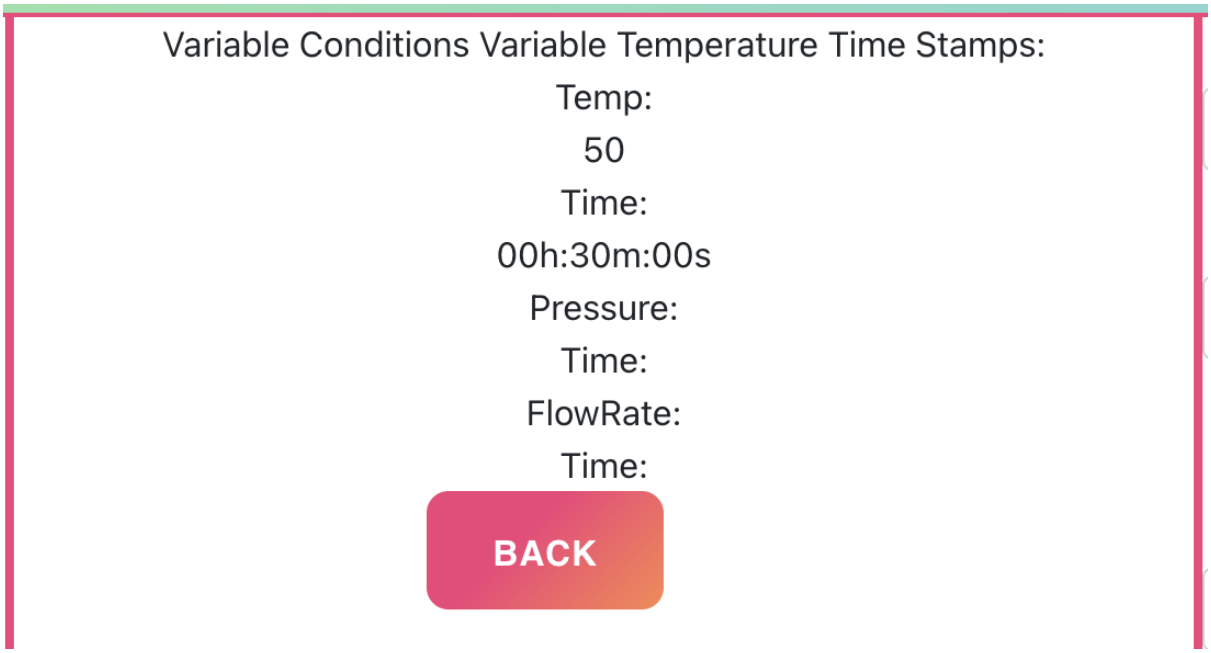

Figure 29: Variable conditions interface

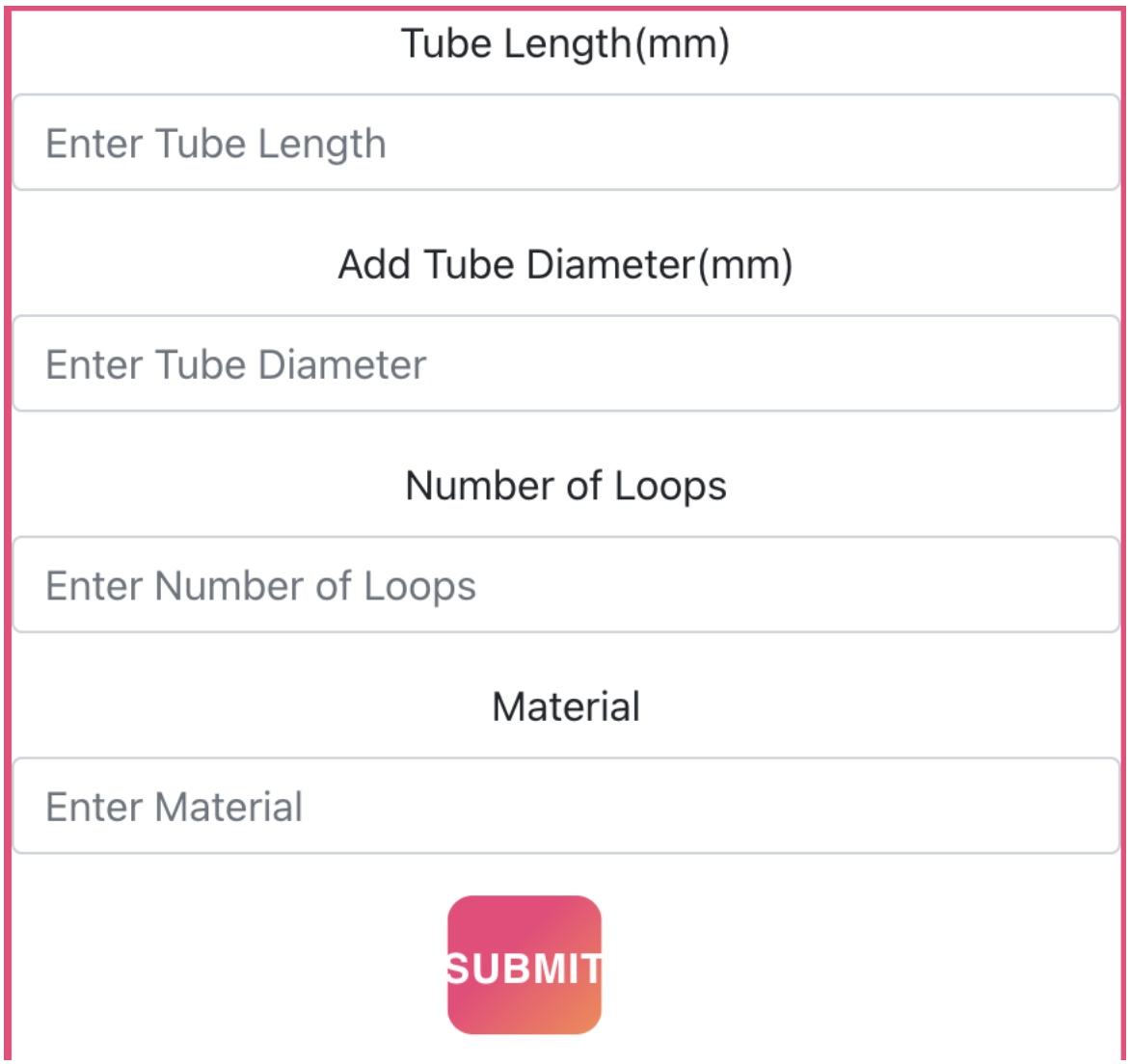

Figure 30: Tube input requirement

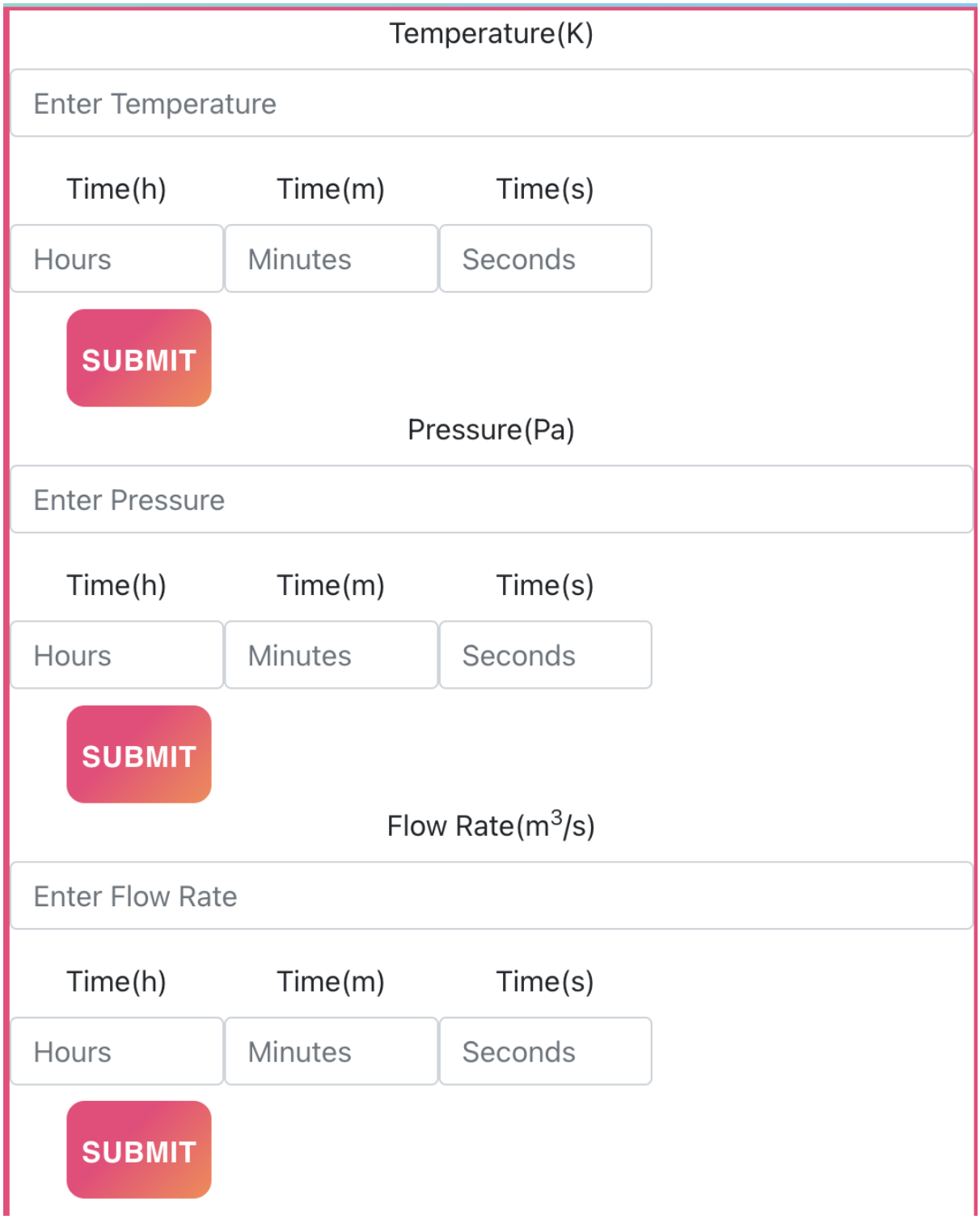

Figure 31: Variable condition input requirements

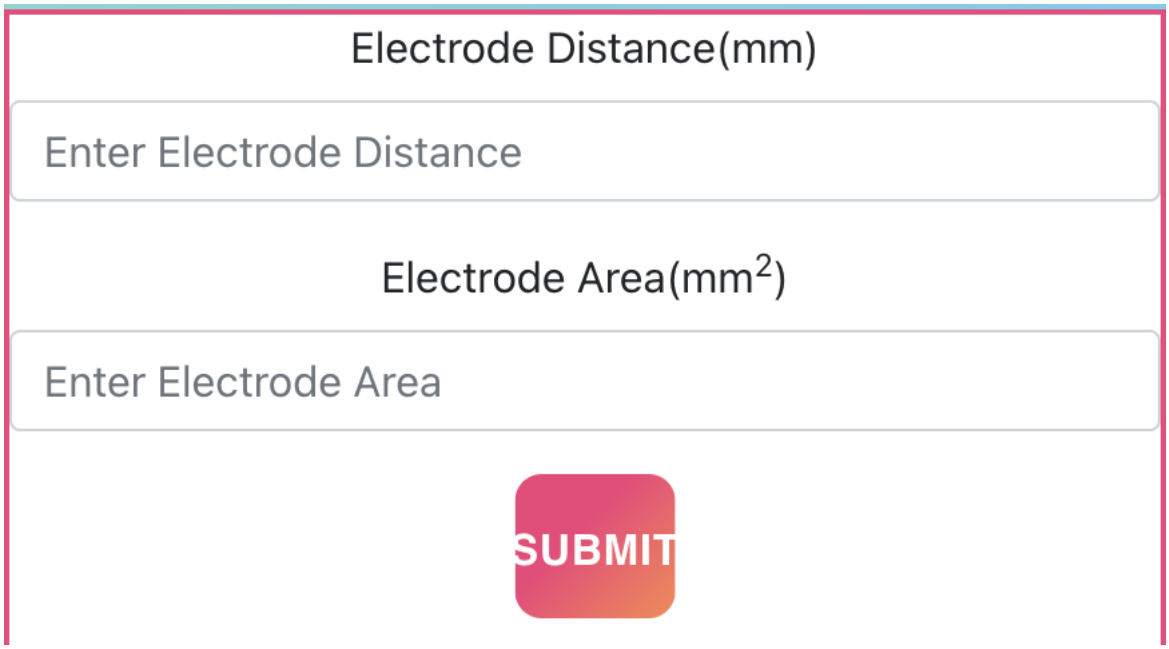

Figure 32: Electrode dimension input requirements

Benzene

Toluene

Methylene

Iron^{3+}

SUBMIT REAGENT ONE

Benzene

Toluene

Methylene

Iron^{3+}

SUBMIT REAGENT TWO

Figure 33: Reagent input requirements

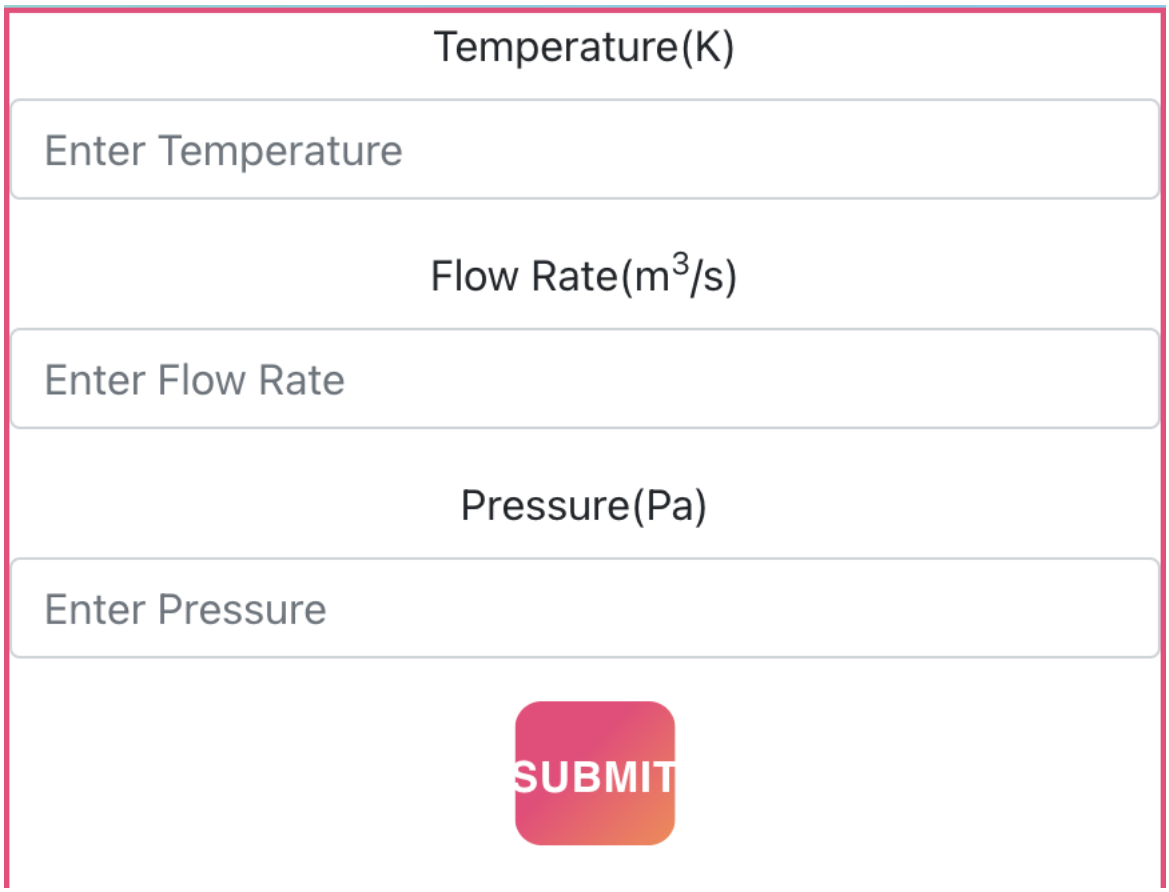

Figure 34: Reaction condition input requirements

It is worth noting that if there are any error messages upon saving a microreactor instance they will be displayed in column 3 replacing whatever component was last rendered there.

The application supports mobile view as shown in figure 35. The columns 1, 2 and 3 are now displayed in a vertical grid comprising of one column per row, with the navigation menu being replaced with a hamburger button.

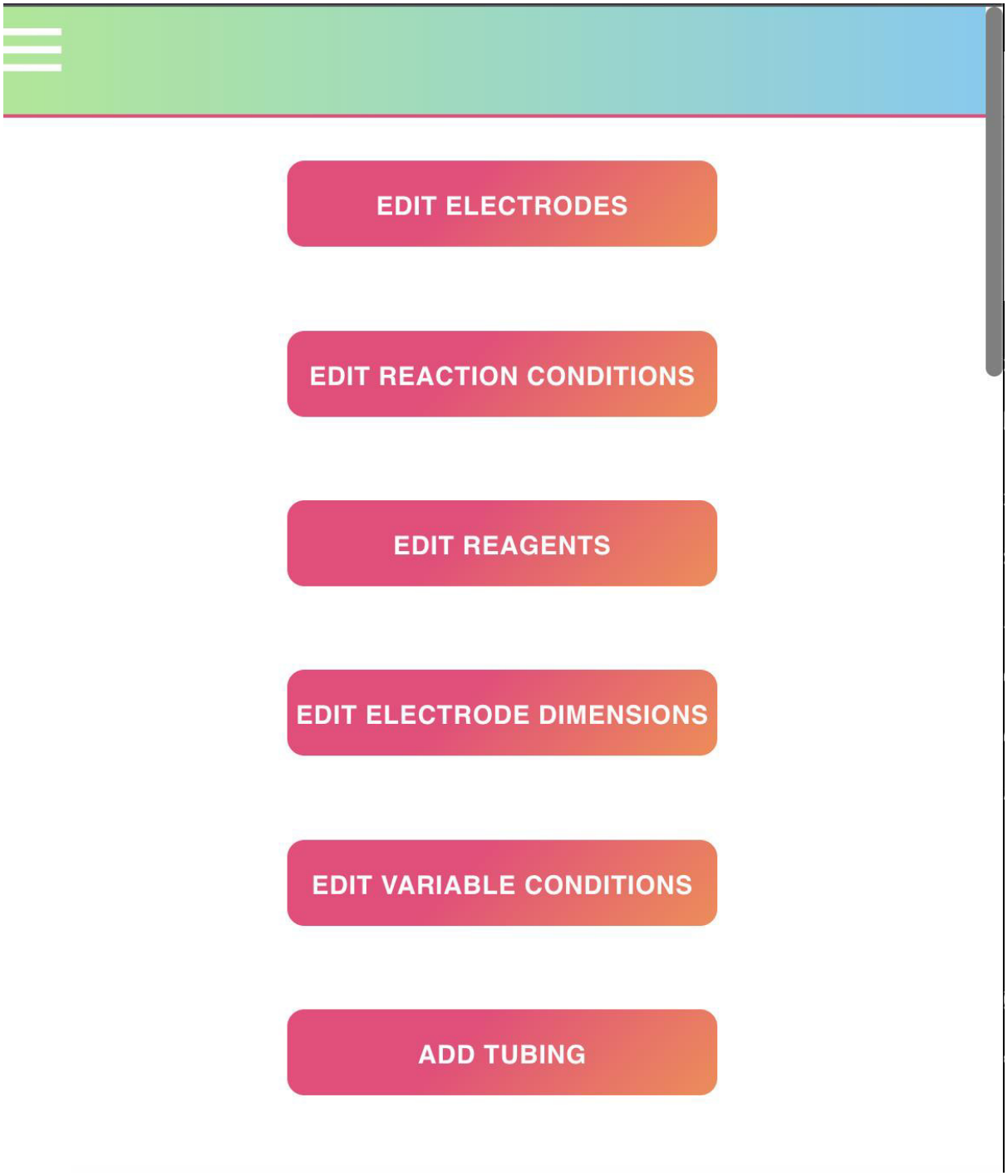

Figure 35: Mobile view with hamburger icon

Hardware Interfaces

Any hardware interfaces are TBD once authorisation for third party software has been granted alongside software update.

Software Interfaces

Connections within the product:

Shown in figure 36 is a context diagram giving a high-level overview as to the applications main function, creating and saving a microreactor instance to the OPC UA server. Shown in the grey box is the physical microreactor which is to be connected after the physical microreactor software has been updated to R-series with 3rd party software authorisation given.

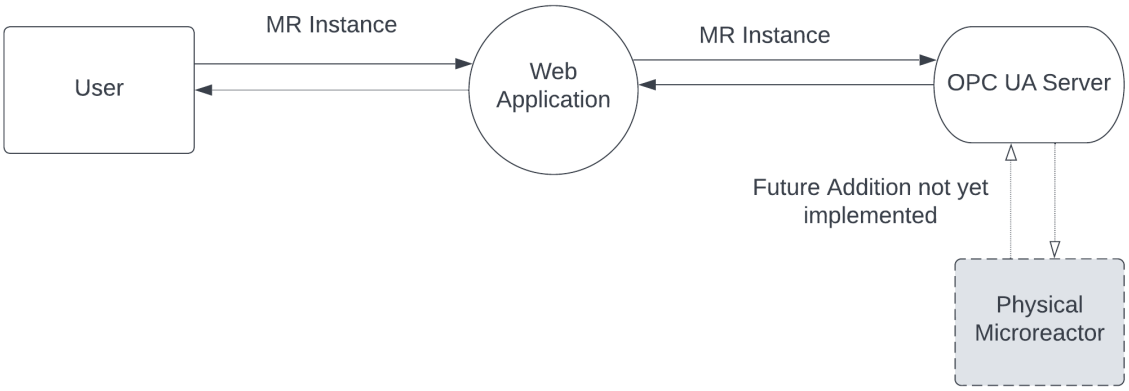

Figure 36: Context diagram of system architecture

Figure 36 shows a high-level dataflow diagram regarding the authentication system of the user. The corresponding login and register functionalities work by using the useEffect object in react.

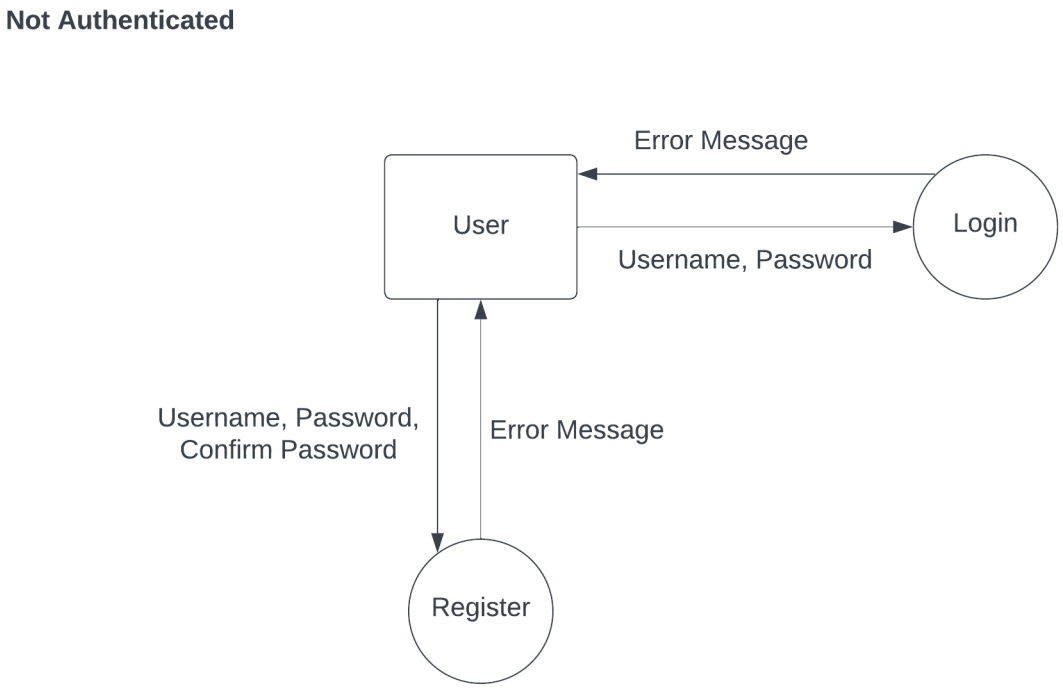

Figure 37: Level 1 data flow diagram (not authenticated)

Figure 38 shows a high-level dataflow diagram regarding the save microreactor function which is explained in the response/stimulus section. As seen in the context diagram, the physical microreactor is an addition to be added in the future once authorisation is granted upon upgrading the software. It is worth noting error messages are only sent back if the specific criterion of the operation isn't met. All operations minus login authentication are handled in the express JS backend which uses axios to send and retrieve data to mongoDB. Data is sent from the OPC UA server back to the user via the test OPC UA connection function which is explained in more detail in section 2.1 in the response/stimulus section. OPC UA server information is displayed to the user via the terminal shell.

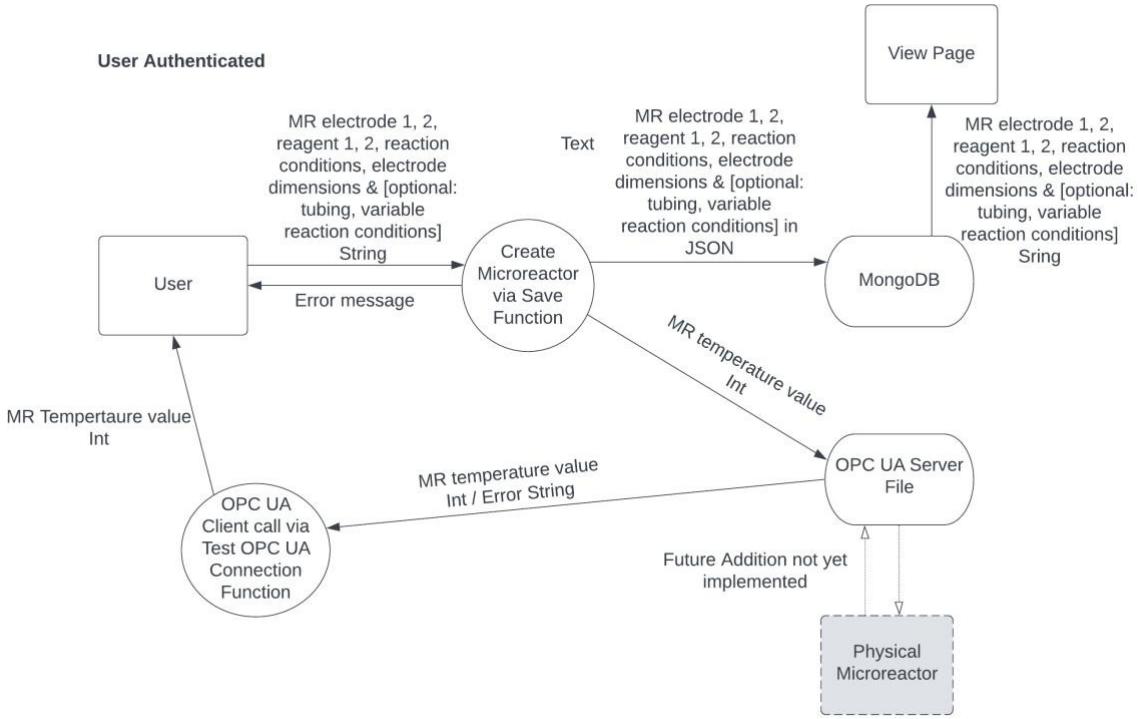

Figure 38: Level 2 data flow diagram (user authenticated)

Communications Interfaces

The OPC UA server despite being locally bound to a personal computer for the time being, is accessible by any user using the web application, as long as the server is running. As this is a web application the standard communication protocol to be used is HTTPS. Future additions to the software will allocate the OPC UA server to be run in a different more accessible location.

Other Nonfunctional Requirements

Safety Requirements

This software utilises the file system API in javascript executed in expressJS which reads and writes files that correlate to the relative file path ../../sample_client_ts/sample_client.js as well

as ../../myserver/sample_server.js. It is important that no such filenames exist in the respective directories upon using the application to avoid alteration of these files.

Security Requirements and Issues

The application will utilise HTTPS once deployed to allow for more security. Further precautions can be taken to ensure more security of the application such as using JWT tokens for account creation and using middleware to further encrypt account data rather than storing some of this data in the local storage in the front-end.

Results and Evaluation

The aim of the project was initially to create a digital twin application that interfaced with a physical Vapourtec R-series microreactor. After, it was learned that it would not be possible to carry out this implementation until the software on the Vapourtec flow system was updated from the flow commander software to the R-series software hence the aim of the project needed to adapt. Consequently, the new aim of the project was to create a digital twin application that interfaced with OPC UA to simulate a digital twin environment until the physical Vapourtec flow system software was updated to the R-series.

Criteria needed to satisfy the aim.

In order to satisfy the aim, OPC UA integration needs to have been proved successful within the application. An OPC UA server environment must be running where the application can send and receive data from it using the web interface, and it is this information exchange that fulfils the definition of a supervisory digital twin. Listed below are the requirements that will need to have been fulfilled in order to satisfy the aim:

- OPC UA server initialisation
- Creation of microreactor object via the web interface
- Data regarding the instantiated microreactor object sent to the OPC UA server via the web application
- OPC UA client-server communication via the web application
- Functional web application (user authentication, navigation, etc…)

To ensure the application has satisfied the aim of the project the criteria above must be fulfilled. The test with Test Case Id TC1 shown below tests the most important functionality of the application: OPC UA integration. For the test to be successful the OPC UA server must be initiated, the OPC UA client call must be successful, and the data it is trying to retrieve must exist thus testing several aspects of OPC UA integration withing the application at once. The code for the test was shown in figure 18 in the response/stimulus section. TC2-TC7 test the general functionality of the web application.

Test Case Id	Summary	Precondition	Steps	Expected Result	Results
TC1	Test OPC UA	Microreactor needs to be created via save button on create page	Click Test OPC UA button	Terminal read of the temperature value submitted during the creation process (Figure 39)	passed
TC2	Account Creation	/	Enter relevant information in Sign up screen shown in figure 15	Redirected to the main authentication screen with the success message “Account created”	passed
TC3	Sign In	Information entered must correlate to that of an already created account	Enter relevant information in Sign in screen shown in figure 16	Redirected to the home page, showing “welcome <username>”	passed
TC4	Sign Out	User must be signed in	Click “Sign out” button	Redirected to the	passed

				authentication screen	
TC5	Login as guest	/	User must click on “Login as guest” tag found on the authentication screen	Redirected to the home page showing “Welcome guest”	passed
TC6	Creation of Microreactor	User must be signed in, and have fulfilled the criteria of the create microreactor function, (more info found in the response/stimulus section)	User must click on “create” in the navigation bar and fill out the data regarding electrodes, reagents, reaction conditions, electrode dimensions and then click the save button	Redirected to the view page with the newly created microreactor displayed under “Saved microreactors”	passed
TC7	Delete Microreactor	User must be signed in and have created a microreactor	Click on the “Delete” button found on the view page, next to any created microreactor	Microreactor removed from the view page, and from mongoDB (refresh required)	passed

Table 1: Test cases

```
connected !
session created !
 value =  { /* DataValue */
   value: Variant(Scalar<Double>, value: 20)
   statusCode:      Good (0x00000000)
   serverTimestamp: 2022-09-29T21:15:34.701Z $ 869.649.000
   sourceTimestamp: 2022-09-29T21:15:34.701Z $ 211.998.000
}
done
```

Figure 39: Terminal response after successful OPC UA client call and data read

Summary of testing results:

The application developed proved to be successful during the testing phase demonstrating a functional web-based application with OPC UA integration, thus fulfilling the criteria of a type 1 digital twin. TC1 demonstrates the application being able to trigger an OPC UA client call to retrieve data from the OPC UA server displaying that data in the terminal as shown in above in figure 39 with the value of 20 representing the Temperature value stored on the OPC UA server corresponding to the newly created microreactor instance that was created in TC6 where the temperature specified was 20. The testing also demonstrates the applications general functionality with respect to navigation, microreactor creation/deletion and authentication system.

Future testing results:

While the software passed all the tests, more tests could have been created to catch less conspicuous bugs. It is important to develop a thorough testing system to ensure the complete functionality of the application. It is also important to take into account the type of tests that were performed too, being manual or automated.

TC2-TC7 relied on manual testing while TC1 demonstrated an example of automated testing. Automated testing utilises scripts to automate testing efforts. The respective script for TC1 is shown at figure 18. TC2-TC7 relied on manual testing meaning that the tests were executed step by step manually without test scripts. Automated testing provides numerous benefits over manual testing such as fewer manual tasks during the testing stage and more test coverage, essentially more testing in less time.

With more time, it would have been beneficial to automate all the tests that were conducted as well as develop more thorough tests to ensure the complete functionality of the application. Such tests could include:

- Testing the application through different screen sizes
- Testing the users access to private/public pages depending on if they're signed in or not
- Individual tests for OPC UA server initialisation, OPC UA client calls, OPC UA server data being changed respectively as opposed to all functionalities being tested simultaneously (TC1)

Project Appraisal:

Considering the results, the software proved to be successful in fulfilling the criteria of a digital twin providing a good foundation to build upon in the future with respect to interfacing with the physical microreactor system. The software however will need to be further improved before moving onto this stage. Additional functionalities to the application will need to be added as the application in its current state fails to initialise the OPC UA server with all respective information pertaining to the microreactor object, only sending the temperature data value as opposed to all the attributes that encompass the microreactor object. The tests carried out took this into account and therefore were somewhat rudimentary and would need to be revised in the future to account for this additional functionality. The next section covers further improvements that the software would benefit from before moving into the next phase of development interfacing with the physical microreactor system.

Future Work

The application created is a type 1 digital twin allowing real-time data exchange between the node OPCUA server environment and the MERN stack web application via node OPC UA client calls. While this project has succeeded in creating a digital twin, the long-term aim of this project is to deliver a digital twin that can remotely control the Vapourtec microreactors as well as use data analysis to help accurately predict and explain reaction pathways. Before the application can be taken to a stage 2 digital twin, there will have to be many revisions of the current software to ensure the successful creation of the subsequent digital twin types. Due to the short time period of this project, there have been additional functionalities of the application that could not be implemented which would need to be added before moving onto the type 2 digital twin.

Listed below are some of the key functionalities and additions to the application that will need to be made:

Improved authentication:

The application handles login data through the front end using react JS in the web browser with local storage, leading the software more prone to attacks. Further improvements of the application would have to deal with user authentication in the backend using express JS middleware with JWT tokens to provide encryption protecting the application from potential exploits.

Improved customisation:

While the application has many options to create a digital twin of a physical microreactor system, it is lacking some extra features such as alloy creation when specifying the electrodes. A future addition of this application could allow the user to generate a custom alloy to be used in place of the electrode. A new data type would have to be created to facilitate the alloy creation in a useful way.

Improved OOP model:

The OOP model that was implemented has been beneficial to the application to store and sort data in an efficient manner. In the future the OOP model will need to be revised and will need more depth to consider more variables. The long-term vision of this project beyond the current scope is to create a digital twin which, with AI can handle large amounts of data to make accurate predictions of reaction outcomes and pathways. With increased data being obtained more accurate predictions will be made, and this will need to be reflected in the OOP model.

More seamless OPC UA integration:

The MERN stack client communicates to the OPC UA server by executing back-end api calls that read and write the OPC UA server and client files. While this allows for a connection to be established between all the respective software, it will provide limitations in the future in regard to speed, and stability. Future additions to the software will need to integrate the node OPC UA client within the expressJS back end to directly fire off OPC UA client calls as opposed to reading and writing the files bound to the computer's local directory.

After communicating with Vapourtec midway through the project it was learned that third party control and data exchange would not be possible with the current software that the Vapourtec microreactors at Cardiff Chemistry use. The R-series Vapourtec microreactors currently use flow commander software which would need to be updated to their latest R-series software to allow third party integration which would facilitate remote operation and data exchange allowing the creation of a type 2 digital twin. The R-series software already connects the microreactor to an OPC UA server, of which data is stored into the OPC UA server and accessed via python scripts shown in figure 1. This presents the option of either interfacing with the OPC UA server directly with the third-party application developed in this project or bypassing coding within the OPC UA architecture altogether utilising their custom-built python scripts.

Shown in table 2 is a pros and cons table regarding the use of python scripts vs the current software architecture.

	Pros	Cons
Python Scripts	• Less development time • Already established framework • Accessible learning resources	• Current web application would need to change to a python-based framework • Limited functionality
OPC UA integration	• No limitation regarding functionality • More established based for future development (e.g incorporation of more sensors within the microreactor system)	• More development time • Scarce learning resources

Table 2: Pros and cons of python scripts vs OPC UA integration

Considering the scripts are exclusive to python, the design of the system which has been coded in Javascript using the MERN stack will need to accommodate for integrating these python scripts or alternatively would need to be coded again in either python flask or more suitably python Django.

Conclusion:

The aim of this project was initially to create a digital twin that could interface with the R-series vapourtec microreactors found in the Cardiff Chemistry department. After it was known that this would not be possible as a result of the software being outdated in flow commander. The aim was then adapted to create a type 1 digital twin that would connect to an OPC UA server and could allow data exchange between both the web client and server. The new aim was established since the vapourtec flow systems utilise OPC UA to allow data exchange between the physical flow system and the R-series software. Creating a digital twin system that interfaces with an OPC UA server would allow for a more seamless transition with respect to future software advancements when interfacing with the physical microreactor system.

Before the specification and design of the software was carried out, a more in depth look into the vapourtec microreactor system, OPC UA, and classification of different types of digital twin were carried out shown in the background section, followed by a brief overview of a similar digital twin system created for a pumping system. Understanding these technologies gave a clear idea as to what is to be expected when creating a digital twin and thus the design and specification process could then begin.

using an object orientated approach to break down the microreactor system so that it could be specified by the user in the web application.

With the aid of the SRS document, a web application was created that interfaced with an OPC UA server using the MERN stack and node OPC UA client respectively. The developed software successfully demonstrated its interoperability between both software, passing the automated test TC1 which entailed OPC UA server initialisation, OPC UA client-server

communication via the web application, and sending data regarding the specified virtual microreactor to the OPC UA server demonstrating full OPC UA integration.

By demonstrating this interoperability, the criteria for a supervisory digital twin as well the project aim was fulfilled, paving the foundation for future advancements to be made to the software that would allow it to interface with the physical microreactor systems found in the Cardiff University Chemistry Department.

It was noted that while the software developed satisfied the aim, the software was still relatively rudimentary in terms of capability and further advancements would need to be made before the next step of development in interfacing with the physical microreactor system can begin. Suggestions for these future advancements were mentioned in the future work section, as well as the results of the testing section.

In a world becoming increasingly conscious of its energy usage, increasing our efficiency in all industrial processes will be key. While this project has shown an insight into the first steps of creating of a digital twin for a microreactor, digital twins will become ubiquitous amongst industry and research in order to help streamline industrial processes and will be one of many steps in helping us live in a more efficient world.

Reflection

At the start of the project, I was assigned the task of creating a digital twin for a microreactor in the Summer of 2021. During this time, remote learning was still in effect. I had no prior experience as to what a digital twin was nor a microreactor despite my undergraduate Chemistry background. After researching the respective topics, I began to develop what I believed to be a digital twin application using python and tkinter. The application allowed users to specify a virtual microreactor with the intention of creating a microreactor object that encapsulated all the attributes of any physical microreactor system. The program did not suffice due to their being no integration with the physical microreactor and therefore failing the criteria as to what constitutes as a digital twin. A revamp of the software was needed that would allow the digital twin software to integrate with the physical microreactor which would not be practical using tkinter as tkinter cannot be deployed as a web-based application rendering future remote usage of the microreactor impossible. I decided to learn the MERN stack due to its scalability for future improvements to the software, its web-based nature as well as it's homogeneity regarding its scripting language, Javascript. After I had coded the basic features of the application, I reached out to Vapourtec who told me that it would be impossible to interface my software with their flow reactor system unless the software that was currently used in the Cardiff laboratories was updated which they offered a large quote for. Time was beginning to run short, and it was made known to me that the software would not be updated during the duration of my project, so I was advised to integrate my application with an OPC UA server which is what the Vapourtec flow systems use to enable machine-machine communication. Information regarding OPC UA was scarce with a lack of physical examples and tutorials provided (particularly for node OPC UA) so it proved troublesome understanding the principles of the language. In the end while the OPC UA integration was successful, it

wasn't ideal. The integration relies on back-end API calls to rewrite local files, which although works, is more prone to errors, could impact communication speeds and is harder to set up.

I had many setbacks throughout the course of the project which had at times felt demoralising, the most recent being the deletion of my entire thesis 17 days before the deadline. I have found that most of the time spent on this project has been learning software as opposed to writing it and there were times where I didn't think the software would be possible to build in the time I had. The combination of scarce materials particularly regarding node OPC UA as well as still being remotely based proved to be a somewhat of an isolating experience for me. I have had to be resilient and stay focussed during this time of setbacks, I appreciate that when things don't go the way we want, we have the option to grow, through the hardship of overcoming or give up.

In retrospect, the sequence of events and setbacks could've been somewhat mitigated through having a better understanding of the scope of the project. A large portion of the problems arose because of early assumptions that were made that resulted in not being true.

The falsely made assumptions were as follows: Tkinter would be a viable language for creating a digital twin, the Vapourtec flow system would be ready to interface with third party software, node OPC UA client calls will be able to be made directly in the express JS backend.

It is important to understand why the aforementioned assumptions were made to identify what can be improved going forward:

Tkinter a viable language for digital twin software: Tkinter, a python module, was chosen due to its abundant learning resources and simple nature thus being straightforward to learn. Python being an object orientated programming language was also taken into consideration at the time due to the nature of the microreactor object have many attributes. It was later learned that the

software couldn't be deployed and thus wouldn't be accessible to establish a connection via the web.

Vapourtec flow system readily available to interface with third party software: This was not taken into consideration until later on during the development process due to a lack of understanding of the scope of the project.

Node OPC UA integration via express JS: The assumption was made due to the node OPC UA client being a node module. Express JS can import node modules and execute them on command. Carrying out these processes caused errors and due to lack of information regarding the node OPC UA client combined with the time left led to the alternative of file manipulation in order to execute node OPC UA client calls.

In summary, the assumptions were made due to a failure to consider all variables combined with an initial lack of understanding of the full scope of the project. Typically, in software development, teams or singular people utilise a development methodology, e.g., the agile or waterfall model before beginning development. This ensures a full mutual understanding of the desired outcome between the developers and the clients. While portions of an agile methodology were used, a scoped-out plan was not fully realised leading to the issues during the development phase.

In the future it will be imperative to adopt a development methodology before starting to implement another project. In the case of this project, while a waterfall method would have solidified a mutual understanding of the desired outcome, there were variables that could not be known until later, (physical Vapourtec flow system integration). While the agile methodology would have been a more viable option, a combination of both methodologies in the blended agile and waterfall model would have been a more suitable methodology approach to this project to ensure a thorough understanding of the desired outcome along with the

flexibility to account for unknown information[11]. This would have reduced the time of completion of the project alongside avoiding many setbacks and is something I will be sure to implement thoroughly when doing any future work.

Appendix

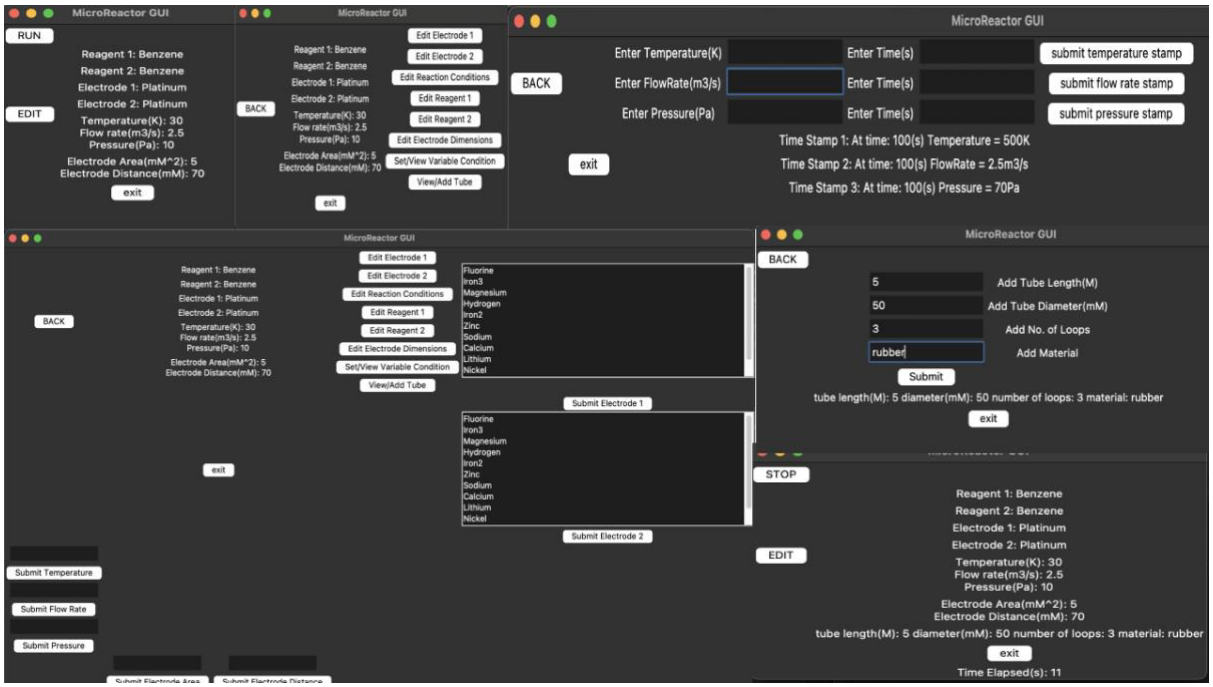

Figure 40: Old design of the virtual microreactor interface

References

1. Negri, E., Fumagalli, L. and Macchi, M., 2017. A Review of the Roles of Digital Twin in CPS-based Production Systems. Procedia Manufacturing, 11, pp.939-948.
2. Oelgemöller, M. and Shvydkiv, O., 2011. Recent Advances in Microflow Photochemistry. Molecules, 16(9), pp.7522-7550.
3. Watanabe, S., Ohsaki, S., Fukuta, A., Hanafusa, T., Takada, K., Tanaka, H., Maki, T., Mae, K. and Miyahara, M., 2017. Characterization of mixing performance in a microreactor and its application to the synthesis of porous coordination polymer particles. Advanced Powder Technology, 28(11), pp.3104-3110.
4. Dong, Z., Wen, Z., Zhao, F., Kuhn, S. and Noël, T., 2021. Scale-up of micro- and milli-reactors: An overview of strategies, design principles and applications. Chemical Engineering Science: X, 10, p.100097.
5. Energy.gov. 2022. Idaho National Laboratory Demonstrates First Digital Twin of a Simulated Microreactor. [online] Available at: <https://www.energy.gov/ne/articles/idaho-national-laboratory-demonstrates-first-digital-twin-simulated-microreactor> [Accessed 28 September 2022].
6. Market, D., 2022. Digital Twin Market Share, Size, Trends - [2022-2027]. [online] MarketsandMarkets. Available at: <https://www.marketsandmarkets.com/Market-Reports/digital-twin-market-225269522.html?gclid=CjwKCAjwp9qZBhBkEiwAsYFsbyGODpqaeqYV9NYCgnapyO6H6tyYMucTn9NcxTcbRCvjK5X38TXZ7hoCLEYQAvD_BwE> [Accessed 28 September 2022].

7. Maity, A., Frey, B., Hoskinson, N. and Powers, D., 2020. Electrocatalytic C–N Coupling via Anodically Generated Hypervalent Iodine Intermediates. Journal of the American Chemical Society, 142(11), pp.4990-4995.
8. Domínguez, M., Centeno, M., Martínez T., M., Bobadilla, L., Laguna, Ó. and Odriozola, J., 2021. Current scenario and prospects in manufacture strategies for glass, quartz, polymers and metallic microreactors: A comprehensive review. Chemical Engineering Research and Design, 171, pp.13-35.
9. Sigmaaldrich.com. 2022. [online] Available at: <https://www.sigmaaldrich.com/GB/en/technical-documents/technical-article/chemistry-and-synthesis/reaction-design-and-optimization/microreactor-technology> [Accessed 28 September 2022].
10. Ansys.com. 2022. [online] Available at: <https://www.ansys.com/content/dam/product/systems-embedded-and-integrated/twin-builder/creating-a-digital-twin-for-a-pump-aa-v11-i1.pdf> [Accessed 28 September 2022].
11. Dursun, M. and Goker, N., 2022. Evaluation of Project Management Methodologies Success Factors Using Fuzzy Cognitive Map Method: Waterfall, Agile, And Lean Six Sigma Cases. International Journal of Intelligent Systems and Applications in Engineering, 10(1), pp.35-43.

```
//App.js
import Intro from './components/Intro';
import SignIn from './components/SignIn';
import Home from './components/Home';
import Layout from './components/Layout';
import Create from './components/Create';
import { BrowserRouter as Router, Route, Routes, Navigate, useParams, useNavigate, Outlet } from 'react-router-dom';
import './App.css';
import { useState, useEffect } from 'react';
import { unstable_renderSubtreeIntoContainer } from 'react-dom';
import { type } from '@testing-library/user-event/dist/type';
import 'bootstrap/dist/css/bootstrap.min.css';
import { LocalConvenienceStoreOutlined, SettingsInputAntennaTwoTone } from '@material-ui/icons';

function App() {


  const [users, setUsers] = useState({
    "sebastian": "password",
    "henry": "hoover",
    "guest": "guest",
  });

  const [user, setUser] = useState({username: "", password: ""});
  const [error, setError] = useState("");
  const [successMessage, setSuccessMessage] = useState("");
  const [test, setTest] =useState("");



  const navigate = useNavigate();
  const [isAuth, setIsAuth] = useState(false);

  const Login = details => {
    console.log(details);
    for (const key in users) {
      const value = users[key]

      if(details.username == key) {

        if (details.password == value) {
          setIsAuth(true);
          setError("");
          setUser({username: key, password: value});
          localStorage.setItem("userUsername", details.username);
          localStorage.setItem("userPassword", details.password);
          navigate('/home');
        } else {
          setError("Details do not match");
        }
      }
    }
    if(users[details.username] == null) {
      setError("Username doesn't exist");

    }
  }

  const signUp = details => {
    let u = details.username;
    let p = details.password;
    let cp = details.confirmPassword;
    if (users[details.username] !== undefined) {
      setError("Username taken");
```

```
  } else {
    if (p == cp) {
      let obj = {};
      obj[u] = p;
      console.log(obj + "obj");
      setUsers({...users, [u] : p});

      setSuccessMessage("Account created");
      setError("");
      navigate('/');

    } else {
       setError("Passwords do not match");
      }
    }



}

const errorNull = () => {
  setError("");
}

const successNull = () => {
  setSuccessMessage("");
}

const guestLogin = () => {
  setIsAuth(true);
  setUser({
    username: "guest",
    password: "guest",
  })
  localStorage.setItem("userUsername", "guest");
  localStorage.setItem("userPassword", "guest");
  authenticated = true;
  navigate('/home')
}

const Logout = () => {
  setIsAuth(false);
  setUser({
    username: "",
    password: "",
  })
  localStorage.setItem("userUsername", "");
  localStorage.setItem("userPassword", "");
  authenticated = false;

  navigate('/Login');
}


let authenticated = false
let userTaken = false

useEffect(() => {
  const u = localStorage.getItem("userUsername");
  const p = localStorage.getItem("userPassword");


  if(u !== null || u !== "") {
    setIsAuth(true);
    setUsers({...users, [u] : p});
    setUser({
```

```
      username: u,
      password: p});
    authenticated = true;
  }

  if(authenticated == false) {
    setIsAuth(false);
  }


 }, []

 )


 return (

  <div className="App">


    {(error !== "") ? (<div className='error'>{error}</div>) : ""}
    {(successMessage !== "") ? (<div className='successMessage'>{successMessage}</div>) : ""}
    <Routes>

      <Route path="" element={isAuth ? <Navigate to ="/Home"/> :
        <Navigate to ="/Login"/>}/>


      <Route path="/*" element={<Layout Logout={Logout} user={user} />} />


      <Route path="/Login" element={<Intro Login={Login} guestLogin={guestLogin} signUp={signUp} successNull =
{successNull} errorNull={errorNull}/>}/>


    </Routes>
  </div>



 );
}

export default App;

//Index.js
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';
import { BrowserRouter as Router} from 'react-router-dom';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
 <React.StrictMode>
  <Router>
    <App />
  </Router>
 </React.StrictMode>
);


reportWebVitals();

//Create.js
```

```
import React from 'react'
import { useState, useEffect } from 'react';
import EditElectrodeOne from './editComponents/EditElectrodeOne';
import EditElectrodeTwo from './editComponents/EditElectrodeTwo';
import EditReactionConditions from './editComponents/EditReactionConditions';
import EditElectrodeDimensions from './editComponents/EditElectrodeDimensions';
import EditReagentOne from './editComponents/EditReagentOne';
import EditReagentTwo from './editComponents/EditReagentTwo';
import EditTubing from './editComponents/EditTubing';
import EditVariableConditions from './editComponents/EditVariableConditions';
import ReactionCol2 from './ReactionCol2';
import VariableConCol2 from './VariableConCol2';
import TubingCol2 from './TubingCol2';
import ShowSaveError from './editComponents/ShowSaveError';
import { Link, useNavigate } from 'react-router-dom';
import Axios from "axios";
import { AlternateEmail, PhotoSizeSelectActualSharp } from '@material-ui/icons';


const Create = () => {




   //button toggle states

   const [showEditElectrodeOne, setShowEditElectrodeOne] = useState(true);
   const [showEditElectrodeTwo, setShowEditElectrodeTwo] = useState(true);
   const [showEditReactionConditions, setShowEditReactionConditions] = useState(false);
   const [showEditReagentOne, setShowEditReagentOne] = useState(false);
   const [showEditReagentTwo, setShowEditReagentTwo] = useState(false);
   const [showEditElectrodeDimensions, setShowEditElectrodeDimensions] = useState(false);
   const [showEditVariableConditions, setShowEditVariableConditions] = useState(false);
   const [showEditAddTubing, setShowEditAddTubing] = useState(false);
   const [showSaveError, setShowSaveError] = useState(false);

   const showList = [showEditElectrodeOne, showEditElectrodeTwo, showEditReactionConditions, showEditReagentOne
      , showEditReagentTwo, showEditElectrodeDimensions, showEditVariableConditions, showEditAddTubing,
showSaveError];

   const setShowList = [setShowEditElectrodeOne, setShowEditElectrodeTwo, setShowEditReactionConditions,
setShowEditReagentOne
      , setShowEditReagentTwo, setShowEditElectrodeDimensions, setShowEditVariableConditions, setShowEditAddTubing,
setShowSaveError];

   //Button toggle functions, for loop turns of all states be default then the if statement turns on the needed component for col 3
   const displayElectrodes = () => {
      for (let i=0; i < showList.length; i++) {
         setShowList[i](false);
      }
      if (showEditElectrodeOne) {
         setShowEditElectrodeOne(false)
      } else {
         setShowEditElectrodeOne(true);
         setShowEditElectrodeTwo(true)};
   }

   const displayReactionConditions = () => {
      for (let i=0; i < showList.length; i++) {
         setShowList[i](false);
      }
      (showEditReactionConditions) ? setShowEditReactionConditions(false) : setShowEditReactionConditions(true);
   }
```

```
const displayReagents = () => {
    for (let i=0; i < showList.length; i++) {
        setShowList[i](false);
    }

    if (showEditReagentOne) {
        setShowEditReagentOne(false)
    } else {
        setShowEditReagentOne(true);
        setShowEditReagentTwo(true)};
}


const displayReagentTwo = () => {
    for (let i=0; i < showList.length; i++) {
        setShowList[i](false);
    }
    (showEditReagentTwo) ? setShowEditReagentTwo(false) : setShowEditReagentTwo(true);
}

const displayElectrodeDimensions = () => {
    for (let i=0; i < showList.length; i++) {
        setShowList[i](false);
    }
    (showEditElectrodeDimensions) ? setShowEditElectrodeDimensions(false) : setShowEditElectrodeDimensions(true);
}

const displayVariableConditions = () => {
    for (let i=0; i < showList.length; i++) {
        setShowList[i](false);
    }
    (showEditVariableConditions) ? setShowEditVariableConditions(false) : setShowEditVariableConditions(true);
}

const displayAddTubing = () => {
    for (let i=0; i < showList.length; i++) {
        setShowList[i](false);
    }
    (showEditAddTubing) ? setShowEditAddTubing(false) : setShowEditAddTubing(true);
}

const displayShowSaveError = () => {
    for (let i=0; i < showList.length; i++) {
        setShowList[i](false);
    }
    setShowSaveError(true);
}

//change center collumn states

const [electrode1, setElectrode1] = useState("");

const [electrode2, setElectrode2] = useState("");

const [reagent1, setReagent1] = useState("");

const [reagent2, setReagent2] = useState("");

const [electrodeDistance, setElectrodeDistance] = useState("");

const [electrodeArea, setElectrodeArea] = useState("");

const [temperature, setTemperature] = useState("");
```

```
const [flowRate, setFlowRate] = useState("");

const [pressure, setPressure] = useState("");

const [temperatureV, setTemperatureV] = useState([]);

const [temperatureVTime, setTemperatureVTime] = useState([]);

const [pressureV, setPressureV] = useState([]);

const [pressureVTime, setPressureVTime] = useState([]);

const [flowRateV, setFlowRateV] = useState([]);

const [flowRateVTime, setFlowRateVTime] = useState([]);

//to render out in hours mins secs lets create 3 variables
const [timeT, setTimeT] = useState([]);

const [timeP, setTimeP] = useState([]);

const [timeF, setTimeF] = useState([]);




const [tubeLength, setTubeLength] = useState([]);

const [tubeDiameter, setTubeDiameter] = useState([]);

const [loops, setLoops] = useState([]);

const [material, setMaterial] = useState([]);

let finalStringVcTemp = "";
let finalStringVcTempTime = "";
let finalStringVcFlowRate = "";
let finalStringVcFlowRateTime = "";
let finalStringVcPressure = "";
let finalStringVcPressureTime;


//function to conve

//this is a string that will be constructed to be stored in the database
const populateVcTemp = () => {
    let stringVcTemp = ''
    //concatatinating all the different variable values in a string but in the format of a list
    for (let i=0; i<temperatureV.length; i++) {
        stringVcTemp = stringVcTemp.concat(`"${temperatureV[i]}", `);
    }
    //removing final space and comma and adding parenthesis
    stringVcTemp = stringVcTemp.slice(0, -2);
    finalStringVcTemp = "[" + stringVcTemp + "]";
    return finalStringVcTemp;
}

const populateVcTempTime = () => {
    let stringVcTempTime = ''
    //concatatinating all the different variable values in a string but in the format of a list
    for (let i=0; i<temperatureVTime.length; i++) {
        stringVcTempTime = stringVcTempTime.concat(`"${temperatureVTime[i]}", `);
    }
    //removing final space and comma and adding parenthesis
    stringVcTempTime = stringVcTempTime.slice(0, -2);
```

```
    finalStringVcTempTime = "[" + stringVcTempTime + "]";
    return finalStringVcTempTime;
  }

  const populateVcPressure = () => {
    let stringVcPressure = ''
    //concatatinating all the different variable values in a string but in the format of a list
    for (let i=0; i<pressureV.length; i++) {
      stringVcPressure = stringVcPressure.concat(`"${pressureV[i]}", `);
    }
    //removing final space and comma and adding parenthesis
    stringVcPressure = stringVcPressure.slice(0, -2);
    finalStringVcPressure = "[" + stringVcPressure + "]";
    return finalStringVcPressure;
  }

  const populateVcPressureTime = () => {
    let stringVcPressureTime = ''
    //concatatinating all the different variable values in a string but in the format of a list
    for (let i=0; i<pressureVTime.length; i++) {
      stringVcPressureTime = stringVcPressureTime.concat(`"${pressureVTime[i]}", `);
    }
    //removing final space and comma and adding parenthesis
    stringVcPressureTime = stringVcPressureTime.slice(0, -2);
    finalStringVcPressureTime = "[" + stringVcPressureTime + "]";
    return finalStringVcPressureTime;
  }

  const populateVcFlowRate = () => {
    let stringVcFlowRate = ''
    //concatatinating all the different variable values in a string but in the format of a list
    for (let i=0; i<flowRateV.length; i++) {
      stringVcFlowRate = stringVcFlowRate.concat(`"${flowRateV[i]}", `);
    }
    //removing final space and comma and adding parenthesis
    stringVcFlowRate = stringVcFlowRate.slice(0, -2);
    finalStringVcFlowRate = "[" + stringVcFlowRate + "]";
    return finalStringVcFlowRate;
  }

  const populateVcFlowRateTime = () => {
    let stringVcFlowRateTime = ''
    //concatatinating all the different variable values in a string but in the format of a list
    for (let i=0; i<flowRateVTime.length; i++) {
      stringVcFlowRateTime = stringVcFlowRateTime.concat(`"${flowRateVTime[i]}", `);
    }
    //removing final space and comma and adding parenthesis
    stringVcFlowRateTime = stringVcFlowRateTime.slice(0, -2);
    finalStringVcFlowRateTime = "[" + stringVcFlowRateTime + "]";
    return finalStringVcFlowRateTime;
  }



  const microReactor = [electrode1, electrode2, temperature, flowRate, pressure, reagent1, reagent2, electrodeDistance,
electrodeArea];
  const microReactorNames = ["Electrode 1", "Electrode 2", "Temperature", "Flow Rate", "Pressure", "Reagent 1", "Reagent
2", "Electrode Distance", "Electrode Height"];




  //changing html center collumn to include sup tags for iron3 and 2


  const [charge, setCharge] = useState("");
```

```
  const [charge2, setCharge2] = useState("");

  // to display the list of set variables in col 2 or to replace it with tubing or variable conditions

  const [col2Reaction, setCol2Reaction] = useState(true);

  const [col2Variable, setCol2Variable] = useState(false);

  const [col2Tubing, setCol2Tubing] = useState(false);



  // functions to change col2 screens

  const col2VariableFunc = () => {
    setCol2Reaction(false);
    setCol2Tubing(false);
    setCol2Variable(true);



  }

  const col2ReactionFunc = () => {
    setCol2Tubing(false);
    setCol2Variable(false);
    setCol2Reaction(true);
  }

  const col2TubingFunc = () => {
    setCol2Reaction(false);
    setCol2Variable(false);
    setCol2Tubing(true);
  }

  let counter = 0;
  const [conditionsSet, setConditionsSet] = useState(counter);

  let counterTube = 0;
  const [conditionsSetTube, setConditionsSetTube] = useState(counterTube);

  const [saveErrorE1, setSaveErrorE1] = useState("");
  const [saveErrorE2, setSaveErrorE2] = useState("");
  const [saveErrorT, setSaveErrorT] = useState("");
  const [saveErrorF, setSaveErrorF] = useState("");
  const [saveErrorP, setSaveErrorP] = useState("");
  const [saveErrorR1, setSaveErrorR1] = useState("");
  const [saveErrorR2, setSaveErrorR2] = useState("");
  const [saveErrorED, setSaveErrorED] = useState("");
  const [saveErrorEA, setSaveErrorEA] = useState("");


  const setSaveErrorList = [setSaveErrorE1, setSaveErrorE2, setSaveErrorT, setSaveErrorF, setSaveErrorP,
setSaveErrorR1, setSaveErrorR2, setSaveErrorED, setSaveErrorEA];

  //setting a state object that will take an array of all saved microreactors (needed to deduce the id number the next microreactor
will be saved as)



    //post microreactor data to back end
```

```
    const [mrCount, setMrCount] = useState(0);



    const mrCountChecker = () => {
        Axios.get("http://localhost:3001/getMicroreactors").then((response) => {

            let newCount = response.data.length + 1;
            setMrCount(newCount);

        });

    }



    useEffect(() => {
        Axios.get("http://localhost:3001/getMicroreactors").then((response) => {
            setMrCount(response.data.length + 1);
        });
    }, []);

    const postMicroreactor = () => {
        //converting all inputted variable condition values to string ready to be stored in database
        let finalStringVcTemp = populateVcTemp();
        let finalStringVcTempTime = populateVcTempTime();
        let finalStringVcPressure = populateVcPressure();
        let finalStringVcPressureTime = populateVcPressureTime();
        let finalStringVcFlowRate = populateVcFlowRate();
        let finalStringVcFlowRateTime = populateVcFlowRateTime();
        let userStored = localStorage.getItem("userUsername");

        Axios.post("http://localhost:3001/postMicroreactor", {
            user: userStored,
            name: mrCount,
            electrodeOne: electrode1,
            electrodeTwo: electrode2,
            temperature: temperature,
            pressure: pressure,
            flowRate: flowRate,
            reagentOne: reagent1,
            reagentTwo: reagent2,
            electrodeDistance: electrodeDistance,
            electrodeArea: electrodeArea,
            vcTemp: finalStringVcTemp,
            vcTempTime: finalStringVcTempTime,
            vcPressure: finalStringVcPressure,
            vcPressureTime: finalStringVcPressureTime,
            vcFlowRate: finalStringVcFlowRate,
            vcFlowRateTime: finalStringVcFlowRateTime,
            //length, diameter, loops, material for tubing
            tubing: [`${tubeLength}, ${tubeDiameter}, ${loops}, ${material}`]



        }).then((response) => {
            console.log(response);
        })};



let navigate = useNavigate();
const save = () => {
    mrCountChecker();
    for(let i=0; i < microReactor.length; i++) {
```

```
            setSaveErrorList[i]("");
            if (microReactor[i] === "") {
                setSaveErrorList[i]("Please submit: " + microReactorNames[i]);
                displayShowSaveError();
            }
        }
        //this counter will only increase if data is submitted, every iteration has data then the function proceeds
        let mrcounterverify = 0;
        for(let i=0; i < microReactor.length; i++) {
            if (microReactor[i] !== "") {
                mrcounterverify ++;
            }

        }

        if (mrcounterverify === 9) {

            postMicroreactor();


            return navigate('/view');
        }


    }


    return (
        <div className='createContainer'>
            <div className='createCol1'>
                <div className='createButton1'>
                    <button onClick={displayElectrodes} className="button-76 introItem1" role="button"><span class="text">Edit
Electrodes</span></button>
                </div>
                <div className='createButton2'>
                    <button onClick={displayReactionConditions} className="button-76 introItem1" role="button"><span
class="text">Edit Reaction Conditions</span></button>
                </div>
                <div className='createButton3'>
                    <button onClick={displayReagents} className="button-76 introItem1" role="button"><span class="text">Edit
Reagents</span></button>
                </div>
                <div className='createButton4'>
                    <button onClick={displayElectrodeDimensions} className="button-76 introItem1" role="button"><span
class="text">Edit Electrode Dimensions</span></button>
                </div>
                <div className='createButton5'>
                    <button onClick={displayVariableConditions} className="button-76 introItem1" role="button"><span
class="text">Edit Variable Conditions</span></button>
                </div>
                <div className='createButton6'>
                    <button onClick={displayAddTubing} className="button-76 introItem1" role="button"><span class="text">Add
Tubing</span></button>
                </div>
                <div className='createButton7'>
                    <button onClick={save} className="button-76 introItem1" role="button"><span class="text">Save</span>
</button>
                </div>
            </div>
            <div className='createCol2'>
                {(col2Reaction && <ReactionCol2 electrode1={electrode1} charge={charge} electrode2={electrode2}
charge2={charge2} temperature={temperature} flowRate={flowRate} pressure={pressure}
                    reagent1={reagent1} reagent2={reagent2} electrodeDistance={electrodeDistance} electrodeArea={electrodeArea}
col2VariableFunc={col2VariableFunc} col2TubingFunc={col2TubingFunc} conditionsSet={conditionsSet}
conditionsSetTube={conditionsSetTube}/>)}
                {(col2Variable && <VariableConCol2 col2ReactionFunc={col2ReactionFunc} temperatureV={temperatureV}
```

```
temperatureVTime={temperatureVTime} pressureV={pressureV} pressureVTime={pressureVTime} flowRateV={flowRateV}
flowRateVTime={flowRateVTime} timeT={timeT} timeF={timeF} timeP={timeP}/>)}
            {(col2Tubing && <TubingCol2 col2ReactionFunc={col2ReactionFunc} tubeLength={tubeLength}
tubeDiameter={tubeDiameter} loops={loops} material={material}/>)}

        </div>
        <div className='createCol3'>
            {(showEditElectrodeOne && <EditElectrodeOne electrode1={electrode1} setElectrode1={setElectrode1}
charge={charge} setCharge={setCharge}/>)}
            {(showEditElectrodeTwo && <EditElectrodeTwo electrode2={electrode2} setElectrode2={setElectrode2}
charge2={charge2} setCharge2={setCharge2}/>)}
            {(showEditReactionConditions && <EditReactionConditions setTemperature={setTemperature}
setFlowRate={setFlowRate} setPressure={setPressure}/>)}
            {(showEditReagentOne && <EditReagentOne reagent1={reagent1} setReagent1={setReagent1}/>)}
            {(showEditReagentTwo && <EditReagentTwo reagent2={reagent2} setReagent2={setReagent2}/>)}
            {(showEditElectrodeDimensions && <EditElectrodeDimensions setElectrodeDistance={setElectrodeDistance}
setElectrodeArea={setElectrodeArea}/>)}
            {(showEditVariableConditions && <EditVariableConditions temperatureV={temperatureV}
setTemperatureV={setTemperatureV} temperatureVTime={temperatureVTime} setTemperatureVTime = {setTemperatureVTime}
timeT={timeT} setTimeT={setTimeT} timeP={timeP} setTimeP={setTimeP} timeF={timeF} setTimeF={setTimeF}
            pressureV={pressureV} setPressureV={setPressureV} pressureVTime={pressureVTime}
setPressureVTime={setPressureVTime} flowRateV={flowRateV} setFlowRateV={setFlowRateV}
flowRateVTime={flowRateVTime} setFlowRateVTime={setFlowRateVTime} conditionsSet={conditionsSet}
setConditionsSet={setConditionsSet}/>)}
            {(showEditAddTubing && <EditTubing conditionsSetTube={conditionsSetTube}
setConditionsSetTube={setConditionsSetTube} tubeLength={tubeLength} setTubeLength={setTubeLength}
tubeDiameter={tubeDiameter} setTubeDiameter={setTubeDiameter} loops={loops} setLoops={setLoops} material={material}
setMaterial={setMaterial}/>)}
            {(showSaveError && <ShowSaveError saveErrorE1={saveErrorE1} saveErrorE2={saveErrorE2}
saveErrorT={saveErrorT} saveErrorF={saveErrorF} saveErrorP={saveErrorP} saveErrorR1={saveErrorR1}
saveErrorR2={saveErrorR2} saveErrorED={saveErrorED} saveErrorEA={saveErrorEA}/>)}

        </div>

    </div>
  )
}

export default Create

//Buttons.js
import React from 'react';
import { Link } from 'react-router-dom';

const Buttons = ({showSignInBtn, showSignUpBtn, guestLogin}) => {
  return (
    <div className='buttonContainer'>
      <button onClick={showSignInBtn} className="button-75 introItem1" role="button" to="/signIn"><span
class="text">Sign In</span></button>
      <button onClick={showSignUpBtn} className="button-75 introItem2" role="button" to="/signUp"><span
class="text">Sign Up</span></button>
      <a className="introItem3" onClick={guestLogin}>Log in as Guest</a>


    </div>
  )
}

export default Buttons

import React, { useState, useEffect } from 'react';

import Axios from 'axios';


import { Outlet, Link } from "react-router-dom";
```

```
const Home = ({user}) => {

 const [numberOfMicroreactors, setNumberOfMicroreactors] = useState(0);

 const [opcuaState, setOpcuaState] = useState("");



 const testOpcuaClient = () => {
  setOpcuaState("testing...");
  Axios.get("http://localhost:3001/testOpcua").then((resp) => {
    setOpcuaState(resp.data);


  })

 };


 useEffect(() => {
  Axios.get("http://localhost:3001/getMicroreactors").then((response) => {

     let arrayOfMrs = [];

     for(let i=0; i < response.data.length;i++) {
        if (response.data[i].user == localStorage.getItem("userUsername")) {
           arrayOfMrs.push(response.data[i])
        }
     }




     setNumberOfMicroreactors(arrayOfMrs.length);



  });


 }, []);

 return (
  <div className="home homeContainer">



     <div className="welcome home1">



       <div className='welcomeText'><h2 id='homeh2'>Welcome, <span>{user.username}</span></h2></div>

       <div className='welcome2 home2'>Number of microreactors created: {numberOfMicroreactors}</div>

       <div className="home3"><button onClick={testOpcuaClient} className="button-76 introItem1" role="button"><span
class="text">Test OPC UA Connection</span></button></div>
       <div className='home4'>{opcuaState}</div>


     </div>
```

```
    </div>
  )
}

export default Home

//Intro.js
import React from 'react'
import Buttons from './Buttons'
import SignUp from './SignUp'
import SignIn from './SignIn'
import { useState } from 'react'

const Intro = ({Login, guestLogin, signUp, successNull, errorNull}) => {


    const [showButtons, setShowButtons] = useState(true)

    const [showSignIn, setShowSignIn] = useState(false)


    const showSignInBtn = () => {
        if (!showSignIn) {
            setShowSignIn(true)
            setShowButtons(false)
            successNull();
        } else {
            setShowSignIn(false)
        }
    }

    const [showSignUp, setShowSignUp] = useState(false)
    const showSignUpBtn = () => {
        if (!showSignIn) {
            setShowSignUp(true)
            setShowButtons(false)
            successNull();

        } else {
            setShowSignUp(false)
        }
    }


    const backBtn = () => {
        if (showSignIn) {
            setShowSignIn(false)
            setShowButtons(true)

        } else if (showSignUp) {
            setShowSignUp(false)
            setShowButtons(true)
        }
        errorNull();

    }



    return (
        <div>
            <div className='introBox'>
                <div className='titleMicroReactor'>
```

```
            CARDIFF MICROREACTOR
          </div>
          {showButtons && <Buttons showSignInBtn={showSignInBtn} guestLogin={guestLogin}
showSignUpBtn={showSignUpBtn}/>}
          {showSignIn && <SignIn backBtn={backBtn} Login={Login}/>}
          {showSignUp && <SignUp backBtn={backBtn} signUp={signUp}/>}



        </div>
      </div>
  )
}

export default Intro

import React, {useState} from 'react'
import { MenuOutlined } from '@material-ui/icons'
import { Close } from '@material-ui/icons'
import { Link, Outlet } from 'react-router-dom'
import Create from './Create'
import Home from './Home'
import View from './View'
import { BrowserRouter as Router, Route, Routes, Navigate, useParams, useNavigate } from 'react-router-dom';



const Layout = ({Logout, user}) => {
 const [active, setActive] = useState(false)

 const showMenu = () => {
  setActive(!active)
 }

 const signout = () => {
  Logout();
 }

 const submitHandler = (e) => {
  e.preventDefault();
  signout();
}

 return (
  <div className='layoutBorder'>
   <div className='layout'>
    <div className='menu-icon'>
     <MenuOutlined className='menu' onClick={showMenu}/>
    </div>
    <nav className={active ? 'slider active' : 'slider'}>
     <ul>
      <div className='closed'>
       <Close className='close' onClick={showMenu}/>

      </div>
      <li className='homeText'>
       <Link to="/home">Home</Link>
      </li>
      <li className='homeText'>
       <Link to="/create">Create Microreactor</Link>
      </li>
      <li className='homeText'>
       <Link to={'/view'}>View Microreactors</Link>
      </li>
      <li className='homeText'>
       <Link to={'/'} onClick={submitHandler}>Sign Out</Link>
```

```
          </li>
        </ul>
      </nav>
    </div>
    <Outlet/>
    <div>
      <Routes>
        <Route path="/home" element={<Home user={user}/>}/>
        <Route path="/create" element={<Create />}/>
        <Route path="/view" element={<View />}/>

      </Routes>
    </div>
    <Outlet/>
  </div>
 )
}

export default Layout

//ReactionCol2.js

import React from 'react'
import { useState } from 'react';


const ReactionCol2 = ({electrode1, charge, electrode2, charge2, temperature, flowRate, pressure, reagent1, reagent2,
electrodeDistance, electrodeArea, col2VariableFunc, col2TubingFunc, conditionsSet, conditionsSetTube}) => {
  const swapScreenV = (e) => {
    e.preventDefault();
    col2VariableFunc();
  }



  const swapScreenT = (e) => {
    e.preventDefault();
    col2TubingFunc();
  }




  return (
    <div className='rc2container'>
      <h2 className='rc21'>Current Conditions:</h2>
      <h3 className='rc22'>Electrode 1: {electrode1}<sup>{charge}</sup></h3>
      <h3 className='rc23'>Electrode 2: {electrode2}<sup>{charge2}</sup></h3>
      <h3 className='rc24'>Temperature: {temperature}</h3>
      <h3 className='rc25'>Flow Rate: {flowRate}</h3>
      <h3 className='rc26'>Pressure: {pressure}</h3>
      <h3 className='rc27'>Reagent 1: {reagent1}</h3>
      <h3 className='rc28'>Reagent 2: {reagent2}</h3>
      <h3 className='rc29'>Electrode Distance: {electrodeDistance}</h3>
      <h3 className='rc30'>Electrode Area: {electrodeArea}</h3>
      <h3 className='rc31'>Variable Conditions: {conditionsSet} <button className='button-75'
onClick={swapScreenV}>View</button></h3>
      <h3 className='rc32'>Tubing: {conditionsSetTube} <button className='button-75'
onClick={swapScreenT}>View</button></h3>
    </div>
  )
}

export default ReactionCol2

//SignIn.js
```

```
import React from 'react'
import { useState } from 'react';

const SignIn = ({backBtn, Login}) => {
   const [details, setDetails] = useState({username: "", password: ""});

   const submitHandler = e => {
      e.preventDefault();
      Login(details);


   }
 return (
   <form className='signInForm'>
      <div className='signInUser form-group'>
         <label>Username</label>
         <input className='form-control' type='text' placeholder='Enter Username' onChange={e => setDetails({...details,
username: e.target.value})} value={details.name}></input>
      </div>
      <div className='signInPass form-group'>
         <label>Password</label>
         <input className='form-control' type='password' placeholder='Enter Password' onChange={e => setDetails({...details,
password: e.target.value})} value={details.password}></input>
      </div>


      <div className='buttonContainerTwo'>
         <button onClick={backBtn} type="button" className="button-75 buttonsSign1" role="button" to="/signIn"><span
class="text">Back</span></button>
         <button onClick={submitHandler} type="submit" className="button-75 buttonsSign2" role="button" to="/signIn"><span
class="text">Enter</span></button>
      </div>







   </form>
 )
}

export default SignIn

//SignUp.js
import React from 'react'
import { useState } from 'react';

const SignUp = ({backBtn, signUp}) => {
   const [details, setDetails] = useState({username: "", password: "", confirmPassword: ""});

   const submitHandler = e => {
      e.preventDefault();
      signUp(details);
   }

   return (
      <form className='signUpForm'>
         <div className='signUpFormUser form-group'>
            <label>Username</label>
            <input className='form-control' type='text' placeholder='Enter Username' onChange={e => setDetails({...details,
username: e.target.value})} value={details.username}></input>
         </div>
         <div className='signUpPass form-group'>
            <label>Password</label>
            <input className='form-control' type='password' placeholder='Enter Password' onChange={e =>
```

```
setDetails({...details, password: e.target.value})} value={details.password}></input>
          </div>
          <div className='signUpConfirm form-group'>
            <label>Confirm Password</label>
            <input className='form-control' type='password' placeholder='Confirm Password' onChange={e =>
setDetails({...details, confirmPassword: e.target.value})} value={details.confirmPassword}></input>
          </div>
          <div className='buttonContainerTwo'>
            <button onClick={backBtn} type="button" className="button-75 buttonsSign1" role="button" to="/signIn"><span
class="text">Back</span></button>
            <button onClick={submitHandler} type="submit" className="button-75 buttonsSign2" role="button" to="/signIn">
<span class="text">Enter</span></button>
          </div>



        </form>


    )
}

export default SignUp

//TubingCol2.js

import React from 'react'

const TubingCol2 = ({col2ReactionFunc, tubeLength, tubeDiameter, loops, material}) => {

    const swapScreen = (e) => {
        e.preventDefault();
        col2ReactionFunc();
    }

    let renderedOutputTL = tubeLength.map(item => <div>{item}</div>)
    let renderedOutputTD = tubeDiameter.map(item => <div>{item}</div>)
    let renderedOutputL = loops.map(item => <div>{item}</div>)
    let renderedOutputM = material.map(item => <div>{item}</div>)


    return (
        <div>
            Tubing
            <div>
                Tube Length: {renderedOutputTL}
                Tube Diameter: {renderedOutputTD}
                Number of Loops: {renderedOutputL}
                Material: {renderedOutputM}

            </div>

            <button className='button-75' onClick={swapScreen}>Back</button>
        </div>
    )
}

export default TubingCol2

//VariableConCol2.js

import React, { useState } from 'react'


const VariableConCol2 = ({col2ReactionFunc, temperatureV, temperatureVTime, pressureV, pressureVTime, flowRateV,
flowRateVTime, timeT, timeP, timeF}) => {
```

```
  const swapScreen = (e) => {
    e.preventDefault();
    col2ReactionFunc();
  }

  const showTemp = () => {
    for(let i=0; i < temperatureV.length; i++) {
      <div>{temperatureV[i]}</div>
      console.log(i);
      console.log(temperatureV[i]);
    }
    console.log(temperatureV.length);
  }




  //creating a map of new div jsx objects to be displayed to iterate through the state list

  let renderedOutputT = temperatureV.map(item => <div>{item}</div>)
  let renderedOutputTTime = timeT.map(item => <div>{item}</div>)
  let renderedOutputP = pressureV.map(item => <div>{item}</div>)
  let renderedOutputPTime = timeP.map(item => <div>{item}</div>)
  let renderedOutputF = flowRateV.map(item => <div>{item}</div>)
  let renderedOutputFTime = timeF.map(item => <div>{item}</div>)


  return (
    <div>
      Variable Conditions
      Variable Temperature Time Stamps:



      <div>Temp: {renderedOutputT}</div>
      <div>Time: {renderedOutputTTime}</div>
      <div>Pressure: {renderedOutputP}</div>
      <div>Time: {renderedOutputPTime}</div>
      <div>FlowRate: {renderedOutputF}</div>
      <div>Time: {renderedOutputFTime}</div>
      <button className='button-75' onClick={swapScreen}>Back</button>
    </div>
  )
}

export default VariableConCol2

//view.js
import React from 'react';
import { useState, useEffect, useRef} from 'react';
import Axios from 'axios';
import { AccordionSummary } from '@material-ui/core';
import { LocalConvenienceStoreOutlined, RepeatOneSharp, SettingsBackupRestoreOutlined } from '@material-ui/icons';


const View = () => {
  //how parse works
  let w = '["wow", "wowww"]';
  const a = JSON.parse(w);
  // list of micro reactors already created shown in the first row that correlate to session
  const [mrViewList, setMrViewList] = useState([]);

  // list of all microreactors regardless of username
  const [mrViewListSelected, setMrViewListSelected] = useState([]);
```

```
//state showing time shown in the bottom middle collumn
//seconds
const [timers, setTimers] = useState(0);
//tens of seconds
const [timerst, setTimerst] = useState(0);
//mins
const [timerm, setTimerm] = useState(0);
//tens of minutes
const [timermt, setTimermt] = useState(0);
//hours
const [timerh, setTimerh] = useState(0);
//tens of hours
const [timerht, setTimerht] = useState(0);
//turns the timer on and off

const [timerToggle, setTimerToggle] = useState(false);

//switches run button to stop upon pressing

const [buttonStateRun, setButtonStateRun] = useState(true);
const [buttonStateStop, setButtonStateStop] = useState(false);

//state that will hide/show the run/stop buttons upon clicking

const [showStop, setShowStop] = useState(false);

//state that will show the selected microreactors variable conditions that have been qued
const [col3vc, setCol3vc] = useState("");

//states will update the variable conditions qued when a mircroreactor is clicked on

const [col2Temp, setCol2Temp] = useState("");
const [col2TempTime, setCol2TempTime] = useState("");
const [col2Pressure, setCol2Pressure] = useState("");
const [col2PressureTime, setCol2PressureTime] = useState("");
const [col2FlowRate, setCol2FlowRate] = useState("");
const [col2FlowRateTime, setCol2FlowRateTime] = useState("");

const [col2SelectedMr, setCol2SelectedMr] = useState("");
const [col2Electrode1, setCol2Electrode1] = useState("");
const [col2Electrode2, setCol2Electrode2] = useState("");
const [col2RcT, setCol2RcT] = useState("");
const [col2RcP, setCol2RcP] = useState("");
const [col2RcF, setCol2RcF] = useState("");
const [col2Reagent1, setCol2Reagent1] = useState("");
const [col2Reagent2, setCol2Reagent2] = useState("");
const [col2ElectrodeDistance, setCol2ElectrodeDistance] = useState("");
const [col2ElectrodeArea, setCol2ElectrodeArea] = useState("");
const [col2NoTubes, setCol2NoTubes] = useState("");









//states will update the variable conditions that have been executed in col 1

const [col1Temp, setCol1Temp] = useState("");
const [col1TempTime, setCol1TempTime] = useState("");
const [col1Pressure, setCol1Pressure] = useState("");
const [col1PressureTime, setCol1PressureTime] = useState("");
const [col1FlowRate, setCol1FlowRate] = useState("");
const [col1FlowRateTime, setCol1FlowRateTime] = useState("");
```

```
//states that will show the times set in h:m:s format
const [timeT, setTimeT] = useState([]);
const [timeP, setTimeP] = useState([]);
const [timeF, setTimeF] = useState([]);

const [dbTimer, setDbTimer] = useState();
let dbTimerCounter = 1;

//state that shows what button has been clicked
const [microReactorClicked, setMicroReactorClicked] = useState(false);

//error which will render if run has been clicked without microreactor clicked
const [microReactorClickedError, setMicroReactorClickedError] = useState();

const [runButtonClicked, setRunButtonClicked] = useState(false);

const [runButtonClickedError, setRunButtonClickedError] = useState("");









let secs = 0;
let secsTens = 0;
let mins = 0;
let minsTens = 0;
let hours = 0;
let hoursTens = 0;

//instantly increments as set interval will delay by 1s on the first iteration
const instantIncrements = () => {
    secs ++;
}

let tempArray = [];

const timerid = useRef(0);
// timer that starts when the run button is clicked
const startTimer = () => {

    if (microReactorClicked) {
        //sorting out states of what was clicked and not to avoid repeats etc
        setRunButtonClicked(true);
        setMicroReactorClickedError("");
        setDbTimer(0);

        setButtonStateRun(false);
        setButtonStateStop(true);
        //avoids a double delay on first iteration
        instantIncrements();

        tempArray = col2Temp.split(" ");

        timerid.current = setInterval(() => {

            if (secs === 10) {
                secs = 0;
                //fixes a delay on first increment of secsTens
                if (secsTens === 0) {
                    setTimerst(1);
                    secsTens = 1;
                // increments minute by 1 once 60 seconds reached
```

```
            } else if (secsTens === 6) {
                //fixes a delay on the first incremenet of minute
                if (mins === 0) {
                    setTimerm(1);
                    mins = 1;
                //increments tens of minutes after 10 minutes reached
                } else if (mins === 10) {
                    //fixes a delay in the first increment of minuteTens
                    if (minsTens === 0) {
                        setTimermt(1);
                        minsTens = 1;
                    //increments hour by one once 60 minutes reached
                    } else if (minsTens === 6) {
                        minsTens = 0;
                        setTimermt(0);
                        //fixes delay on first hour
                        if (hours === 0) {
                            setTimerh(1);
                            hours = 1;
                        } else if (hours === 10) {
                            hours = 0;
                            setTimerh(0);
                            //fixes delay on first hourTens
                            if(hoursTens === 0) {
                                setTimerht(1);
                                hoursTens = 1;

                            }
                            setTimerht(hoursTens ++);
                        }
                        setTimerh(hours ++);
                    }

                    setTimermt(minsTens ++);

                    mins = 0


                }

                secsTens = 0;
                setTimerm(mins ++);
            }
            setTimerst(secsTens ++);
        }
        setTimers(secs ++);



        dbTimerCounter ++;
    }, 1000);

} else {
    setMicroReactorClickedError("please select a microreactor");
}




}

const startTimerTwo = () => {

}

const stopTimer = () => {
```

```
        setRunButtonClicked(false);
        clearInterval(timerid.current);
        timerid.current = 0;
        setButtonStateRun(true);
        setButtonStateStop(false);
        secs = 0;
        secsTens = 0;
        mins = 0;
        minsTens = 0;
        hours = 0;
        hoursTens = 0;
        setTimers(0);
        setTimerst(0);
        setTimerm(0);
        setTimermt(0);
        setTimerh(0);
        setTimerht(0);
        setCol1Temp("");
        setCol1TempTime("");
        setCol1Pressure("");
        setCol1PressureTime("");
        setCol1FlowRate("");
        setCol1FlowRateTime("");
    }


    //states that will be used when the run button is clicked to check if time condition has been satisfied to then move from col3 to col1
    const [runTemp, setRunTemp] = useState([]);
    const [runTempTime, setRunTempTime] = useState([]);
    const [runPressure, setRunPressure] = useState([]);
    const [runPressureTime, setRunPressureTime] = useState([]);
    const [runFlowRate, setRunFlowRate] = useState([]);
    const [runFlowRateTime, setRunFlowRateTime] = useState([]);

    useEffect(() => {
        Axios.get("http://localhost:3001/getMicroreactors").then((response) => {

            let arrayOfMrs = [];

            for(let i=0; i < response.data.length;i++) {
                if (response.data[i].user == localStorage.getItem("userUsername")) {
                    arrayOfMrs.push(response.data[i])
                }
            }




            setMrViewList(arrayOfMrs);



        });


    }, []);




    const deleteMicroreactor = (id) => {
```

```
    Axios.delete(`http://localhost:3001/delete/${id}`);
    setMrViewList(mrViewList.filter((val) => {
      return val.id != id;

    }))

}

const readOpcuaClient = () => {
    Axios.get("http://localhost:3001/readOpcua").then((resp) => {

    })
};

const runOpcuaClient = () => {
    Axios.get("http://localhost:3001/runOpcua").then((resp) => {

    })
};




const [test1, setTest1] = useState("00:33:22s");
const[test2, setTest2] = useState("");

return (
    <div>
      <div className='viewContainer'>
        <div className='viewCol1'>
          <div className='savedMr'>Saved Microreactors:</div>



          <div className='view3Buttons'>
            {mrViewList.map((mr) => {
              return (
                <div className="viewColumn" key={mr.name} onClick={() => {
                  if (runButtonClicked === false) {
                    setMicroReactorClicked(true);
                    setMicroReactorClickedError("");

                    //setting the bottom middle collumn on click
                    setCol2SelectedMr(mr.name);
                    setCol2Electrode1(mr.electrodeOne);
                    setCol2Electrode2(mr.electrodeTwo);
                    setCol2RcT(mr.temperature);
                    setCol2RcP(mr.pressure);
                    setCol2RcF(mr.flowRate);
                    setCol2Reagent1(mr.reagentOne);
                    setCol2Reagent2(mr.reagentTwo);
                    setCol2ElectrodeDistance(mr.electrodeDistance);
                    setCol2ElectrodeArea(mr.electrodeArea);
                    setCol2NoTubes(mr.tubing);




                    //converting string into list
```

```
                        let listVcTemp = JSON.parse(mr.vcTemp);
                        let listVcTempTime = JSON.parse(mr.vcTempTime);
                        let listVcPressure = JSON.parse(mr.vcPressure);
                        let listVcPressureTime = JSON.parse(mr.vcPressureTime);
                        let listVcFlowRate = JSON.parse(mr.vcFlowRate);
                        let listVcFlowRateTime = JSON.parse(mr.vcFlowRateTime);

                        setRunTemp(listVcTemp);
                        setRunTempTime(listVcTempTime);
                        setRunPressure(listVcPressure);
                        setRunPressureTime(listVcPressureTime);
                        setRunFlowRate(listVcFlowRate);
                        setRunFlowRateTime(listVcFlowRateTime);




                        //e.g col2TempString is a string that will have all the list elements inserted into it from vcTemp in the
database to be displayed in col2
                        let col2TempString = "";
                        let col2TempTimeString = "";
                        let col2PressureString = "";
                        let col2PressureTimeString = "";
                        let col2FlowRateString = "";
                        let col2FlowRateTimeString = "";
                        //e.g concats the data together from the vcTemp
                        for(let i=0; i<listVcTemp.length; i++) {
                            col2TempString = col2TempString.concat(listVcTemp[i] + ", ");
                        }
                        //gets rid of the final ", " in the string and then sets the state to the string
                        col2TempString = col2TempString.slice(0, -2);
                        setCol2Temp(col2TempString);

                        let finalListT = [];
                        //same but for temp time
                        for(let i=0; i<listVcTempTime.length; i++) {
                            //converting to int
                            let temporaryIterationT = parseInt(listVcTempTime[i]);
                            //finding a floored values to get number of hours
                            let hoursT = Math.floor(temporaryIterationT / 3600);
                            //same for mins and seconds respectively
                            let minsT = Math.floor((temporaryIterationT - (hoursT * 3600))/60);
                            let secsT = Math.floor(temporaryIterationT - (hoursT * 3600) - (minsT * 60));
                            //setting states to update the new time format as a formatted string
                            let finalString = `${hoursT}h:${minsT}m:${secsT}s`;
                                finalListT.push(finalString)

                            col2TempTimeString = col2TempTimeString.concat(finalListT[i] + ", ");
                        }




                        col2TempTimeString = col2TempTimeString.slice(0, -2);
                        setCol2TempTime(col2TempTimeString);

                        //same but for pressure
                        for(let i=0; i<listVcPressure.length; i++) {
                            col2PressureString = col2PressureString.concat(listVcPressure[i] + ", ");
                        }

                        col2PressureString = col2PressureString.slice(0, -2);
                        setCol2Pressure(col2PressureString);

                        let finalListP = [];
                        //same but for pressure time
```

```
                        for(let i=0; i<listVcPressureTime.length; i++) {
                            //converting to int
                            let temporaryIterationP = parseInt(listVcPressureTime[i]);
                            //finding a floored values to get number of hours
                            let hoursP = Math.floor(temporaryIterationP / 3600);
                            //same for mins and seconds respectively
                            let minsP = Math.floor((temporaryIterationP - (hoursP * 3600))/60);
                            let secsP = Math.floor(temporaryIterationP - (hoursP * 3600) - (minsP * 60));
                            //setting states to update the new time format as a formatted string
                            let finalString = `${hoursP}h:${minsP}m:${secsP}s`;
                                finalListP.push(finalString)
                            col2PressureTimeString = col2PressureTimeString.concat(finalListP[i] + ", ");
                        }

                        col2PressureTimeString = col2PressureTimeString.slice(0, -2);
                        setCol2PressureTime(col2PressureTimeString);

                        //same but for flowRate
                        for(let i=0; i<listVcFlowRate.length; i++) {
                            col2FlowRateString = col2FlowRateString.concat(listVcFlowRate[i] + ", ");
                        }
                        col2FlowRateString = col2FlowRateString.slice(0, -2);
                        setCol2FlowRate(col2FlowRateString);

                        let finalListF =[];
                        //same but for flowRate Time
                        for(let i=0; i<listVcFlowRateTime.length; i++) {
                            //converting to int
                            let temporaryIterationF = parseInt(listVcFlowRateTime[i]);
                            //finding a floored values to get number of hours
                            let hoursF = Math.floor(temporaryIterationF / 3600);
                            //same for mins and seconds respectively
                            let minsF = Math.floor((temporaryIterationF - (hoursF * 3600))/60);
                            let secsF = Math.floor(temporaryIterationF - (hoursF * 3600) - (minsF * 60));
                            //setting states to update the new time format as a formatted string
                            let finalString = `${hoursF}h:${minsF}m:${secsF}s`;
                                finalListF.push(finalString)
                            col2FlowRateTimeString = col2FlowRateTimeString.concat(finalListF[i] + ", ");
                        }

                        col2FlowRateTimeString = col2FlowRateTimeString.slice(0, -2);
                        setCol2FlowRateTime(col2FlowRateTimeString);
                    }

                    }

                }>
                    <div className='hoverMR viewCol10'>Microreactor number: {mr.name}</div>
                    <div className='viewCol20'><button className="button-77"
onClick={runOpcuaClient}>View</button></div>
                    <div className='viewCol30'><button className="button-77" onClick={() =>
{deleteMicroreactor(mr._id)}}>Delete</button></div>
                </div>
            )
        })}

        </div>
    </div>
    <div className='viewCol2'>
        <div className="viewContainer2">
            <div className='view2Col1'>
                <div className='savedMr'>
                    Variable condition changes executed:
                </div>
                <div>
                    Temperature(K): {col1Temp}
```

```
                </div>
                <div>
                  Time: {col1TempTime}
                </div>
                <div>
                  Pressure(Pa): {col1Pressure}
                </div>
                <div>
                  Time: {col1PressureTime}
                </div>
                <div>
                  FlowRate(m<sup>3</sup>/s): {col1FlowRate}
                </div>
                <div>
                  Time: {col1FlowRateTime}
                </div>

              </div>
              <div className='view2Col2'>
                <div className="viewMid1 savedMr">Time elapsed: </div>
                <div className="viewMid2">{timerht}{timerh}h:{timermt}{timerm}m:{timerst}{timers}s</div>
                <div className="viewMid3">Microreactor selected: {col2SelectedMr}</div>
                <div className="viewMid4">Electrode 1: {col2Electrode1}</div>
                <div className="viewMid5">Electrode2: {col2Electrode2}</div>
                <div className="viewMid6">Temperature(K): {col2RcT}</div>
                <div className="viewMid7">Flow Rate(m<sup>3</sup>/s): {col2RcF}</div>
                <div className="viewMid8">Pressure(Pa): {col2RcP}</div>
                <div className="viewMid9">Reagent 1: {col2Reagent1}</div>
                <div className="viewMid10">Reagent 2: {col2Reagent2}</div>
                <div className="viewMid11">Electrode Distance(mm): {col2ElectrodeDistance}</div>
                <div className="viewMid12">Electrode Area(mm<sup>2</sup>): {col2ElectrodeArea}</div>
                <div className="viewMid13">Tubes: {col2NoTubes}</div>
                <div className="viewMid14">


                {buttonStateRun && <button onClick={startTimer} className="button-75 runButtonAdjust">Run</button>}
                    {buttonStateStop && <button onClick={stopTimer} className="button-75
runButtonAdjust">Stop</button>}

                <div className='savedMr'>{microReactorClickedError}</div>
                <div className='savedMr'>{runButtonClickedError}</div>
                </div>

              </div>
              <div className='view2Col3'>
                <div className='savedMr'>
                  Variable Conditions qued:
                </div>
                <div>
                  Temperature(K): {col2Temp}
                </div>
                <div>
                  Time: {col2TempTime}
                </div>
                <div>
                  Pressure(Pa): {col2Pressure}
                </div>
                <div>
                  Time: {col2PressureTime}
                </div>
                <div>
                  FlowRate(m<sup>3</sup>/s): {col2FlowRate}
                </div>
                <div>
                  Time: {col2FlowRateTime}
                </div>
              </div>
```

```
            </div>
          </div>
        </div>


      </div>
    )
}

export default View

//EditElectrodeDimensions.js
import React, { useState } from 'react'


const EditElectrodeDimensions = ({setElectrodeDistance, setElectrodeArea}) => {

    const [details, setDetails] = useState({electrodeDistance: "", electrodeArea: ""});
    const [detailsED, setDetailsED] = useState("");
    const [detailsEA, setDetailsEA] = useState("");
    const[electrodeDError, setElectrodeDError] = useState("");


    const submitHandler = (e) => {
        e.preventDefault();
        //error prevention /^[0-9]+$/.test() checks all characters in string are digits
        if (/^[0-9]+$/.test(details.electrodeArea) && /^[0-9]+$/.test(details.electrodeDistance)) {
            setElectrodeDistance(details.electrodeDistance);
            setElectrodeArea(details.electrodeArea);
            setElectrodeDError("");

        } else {
            setElectrodeDError("Please enter integer values only for both input fields");
        }

    }



    return (
        <div>
            <form className='signUpForm'>
                <div className='signUpFormUser form-group'>
                    <label>Electrode Distance(mm)</label>
                    <input className='form-control' type='text' placeholder='Enter Electrode Distance' onChange={e =>
setDetails({...details, electrodeDistance: e.target.value})} value={details.electrodeDistance}></input>
                </div>
                <div className='signUpPass form-group'>
                    <label>Electrode Area(mm<sup>2</sup>)</label>
                    <input className='form-control' type='text' placeholder='Enter Electrode Area' onChange={e =>
setDetails({...details, electrodeArea: e.target.value})} value={details.electrodeArea}></input>
                </div>
                <div className='buttonContainerTwo buttonStraightener'>
                    <button type="submit" onClick={submitHandler} className="button-75 buttonsSign2" role="button"><span
class="text">Submit</span></button>
                </div>
                <div>{electrodeDError}</div>
            </form>

        </div>
    )
}

export default EditElectrodeDimensions

//EditElectrodeOne.js
```

```
import React from 'react';
import { useState } from 'react';

const EditElectrodeOne = ({electrode1, setElectrode1, charge, setCharge}) => {

const [changeStyle1, setChangeStyle1] = useState(false);
const [changeStyle2, setChangeStyle2] = useState(false);
const [changeStyle3, setChangeStyle3] = useState(false);
const [changeStyle4, setChangeStyle4] = useState(false);
const [changeStyle5, setChangeStyle5] = useState(false);
const [changeStyle6, setChangeStyle6] = useState(false);
const [changeStyle7, setChangeStyle7] = useState(false);
const [changeStyle8, setChangeStyle8] = useState(false);
const [changeStyle9, setChangeStyle9] = useState(false);
const [changeStyle10, setChangeStyle10] = useState(false);

const stateList = [changeStyle1, changeStyle2, changeStyle3, changeStyle4, changeStyle5, changeStyle6, changeStyle7,
changeStyle8, changeStyle9, changeStyle10];

const stateListFunc = [setChangeStyle1, setChangeStyle2, setChangeStyle3, setChangeStyle4, setChangeStyle5,
setChangeStyle6, setChangeStyle7, setChangeStyle8, setChangeStyle9, setChangeStyle10];

const elements = ["Magnesium", "Strontium", "Fluorine", "Iron3", "Hydrogen", "Iron2", "Sodium", "Calcium", "Lithium", "Nickel"];


const changeLiStyle = () => {
   for(let i = 0; i < stateList.length; i++) {
      stateListFunc[i](false);
   }
   (changeStyle1 ? setChangeStyle1(false) : setChangeStyle1(true));
}

const changeLiStyle2 = () => {
   for(let i = 0; i < stateList.length; i++) {
      stateListFunc[i](false);
   }
   (changeStyle2 ? setChangeStyle2(false) : setChangeStyle2(true));
}

const changeLiStyle3 = () => {
   for(let i = 0; i < stateList.length; i++) {
      stateListFunc[i](false);
   }
   (changeStyle3 ? setChangeStyle3(false) : setChangeStyle3(true));
}

const changeLiStyle4 = () => {
   for(let i = 0; i < stateList.length; i++) {
      stateListFunc[i](false);
   }
   (changeStyle4 ? setChangeStyle4(false) : setChangeStyle4(true));
}

const changeLiStyle5 = () => {
   for(let i = 0; i < stateList.length; i++) {
      stateListFunc[i](false);
   }
   (changeStyle5 ? setChangeStyle5(false) : setChangeStyle5(true));
}

const changeLiStyle6 = () => {
   for(let i = 0; i < stateList.length; i++) {
      stateListFunc[i](false);
   }
   (changeStyle6 ? setChangeStyle6(false) : setChangeStyle6(true));
```

```
}

const changeLiStyle7 = () => {
   for(let i = 0; i < stateList.length; i++) {
      stateListFunc[i](false);
   }
   (changeStyle7 ? setChangeStyle7(false) : setChangeStyle7(true));
}

const changeLiStyle8 = () => {
   for(let i = 0; i < stateList.length; i++) {
      stateListFunc[i](false);
   }
   (changeStyle8 ? setChangeStyle8(false) : setChangeStyle8(true));
}

const changeLiStyle9 = () => {
   for(let i = 0; i < stateList.length; i++) {
      stateListFunc[i](false);
   }
   (changeStyle9 ? setChangeStyle9(false) : setChangeStyle9(true));
}

const changeLiStyle10 = () => {
   for(let i = 0; i < stateList.length; i++) {
      stateListFunc[i](false);
   }
   (changeStyle10 ? setChangeStyle10(false) : setChangeStyle10(true));
}

// changes state for center collumn and uses charge state to use in sup tag for the html presentation
const submitHandler = () => {
   for(let i = 0; i < stateList.length; i++) {
      if(stateList[i] === true) {
         setElectrode1(elements[i]);
         if(elements[i] === "Iron3") {
            setCharge("3+");
            setElectrode1("Iron");
            console.log("Pause");
         } else if (elements[i] === "Iron2"){
            setCharge("2+");
            setElectrode1("Iron");

         } else {
            setElectrode1(elements[i]);
            setCharge("");
         }
      }
   }
}









   return (
      <div>
         <ul className='editElectrodeBox'>
            <li className={changeStyle1 ? 'electrodeBoxItem selected' : 'electrodeBoxItem'} onClick={changeLiStyle}
id='elementOne'>Magnesium</li>
            <li className={changeStyle2 ? 'electrodeBoxItem selected' : 'electrodeBoxItem'} onClick={changeLiStyle2}
id='2'>Strontium</li>
            <li className={changeStyle3 ? 'electrodeBoxItem selected' : 'electrodeBoxItem'} onClick={changeLiStyle3}
id='3'>Flurorine</li>
```

```
            <li className={changeStyle4 ? 'electrodeBoxItem selected' : 'electrodeBoxItem'} onClick={changeLiStyle4}
id='4'>Iron<sup>3+</sup></li>
            <li className={changeStyle5 ? 'electrodeBoxItem selected' : 'electrodeBoxItem'} onClick={changeLiStyle5}
id='5'>Hydrogen</li>
            <li className={changeStyle6 ? 'electrodeBoxItem selected' : 'electrodeBoxItem'}
onClick={changeLiStyle6}id='6'>Iron<sup>2+</sup></li>
            <li className={changeStyle7 ? 'electrodeBoxItem selected' : 'electrodeBoxItem'} onClick={changeLiStyle7}
id='7'>Sodium</li>
            <li className={changeStyle8 ? 'electrodeBoxItem selected' : 'electrodeBoxItem'} onClick={changeLiStyle8}
id='8'>Calcium</li>
            <li className={changeStyle9 ? 'electrodeBoxItem selected' : 'electrodeBoxItem'} onClick={changeLiStyle9}
id='9'>Lithium</li>
            <li className={changeStyle10 ? 'electrodeBoxItem selected' : 'electrodeBoxItem'} onClick={changeLiStyle10}
id='10'>Nickel</li>
          </ul>
          <div className='electrode1submit'><button className='button-75' onClick={submitHandler}>Submit Electrode
1</button></div>
        </div>
    )
}


export default EditElectrodeOne
//EditElectrodeTwo.js
import React from 'react';
import { useState } from 'react';

const EditElectrodeTwo = ({electrode2, setElectrode2, charge2, setCharge2}) => {

const [changeStyle1, setChangeStyle1] = useState(false);
const [changeStyle2, setChangeStyle2] = useState(false);
const [changeStyle3, setChangeStyle3] = useState(false);
const [changeStyle4, setChangeStyle4] = useState(false);
const [changeStyle5, setChangeStyle5] = useState(false);
const [changeStyle6, setChangeStyle6] = useState(false);
const [changeStyle7, setChangeStyle7] = useState(false);
const [changeStyle8, setChangeStyle8] = useState(false);
const [changeStyle9, setChangeStyle9] = useState(false);
const [changeStyle10, setChangeStyle10] = useState(false);

const stateList = [changeStyle1, changeStyle2, changeStyle3, changeStyle4, changeStyle5, changeStyle6, changeStyle7,
changeStyle8, changeStyle9, changeStyle10];

const stateListFunc = [setChangeStyle1, setChangeStyle2, setChangeStyle3, setChangeStyle4, setChangeStyle5,
setChangeStyle6, setChangeStyle7, setChangeStyle8, setChangeStyle9, setChangeStyle10];

const elements = ["Magnesium", "Strontium", "Fluorine", "Iron3", "Hydrogen", "Iron2", "Sodium", "Calcium", "Lithium", "Nickel"];


const changeLiStyle = () => {
    for(let i = 0; i < stateList.length; i++) {
        stateListFunc[i](false);
    }
    (changeStyle1 ? setChangeStyle1(false) : setChangeStyle1(true));
}

const changeLiStyle2 = () => {
    for(let i = 0; i < stateList.length; i++) {
        stateListFunc[i](false);
    }
    (changeStyle2 ? setChangeStyle2(false) : setChangeStyle2(true));
}

const changeLiStyle3 = () => {
    for(let i = 0; i < stateList.length; i++) {
```

```
        stateListFunc[i](false);
    }
    (changeStyle3 ? setChangeStyle3(false) : setChangeStyle3(true));
}

const changeLiStyle4 = () => {
    for(let i = 0; i < stateList.length; i++) {
        stateListFunc[i](false);
    }
    (changeStyle4 ? setChangeStyle4(false) : setChangeStyle4(true));
}

const changeLiStyle5 = () => {
    for(let i = 0; i < stateList.length; i++) {
        stateListFunc[i](false);
    }
    (changeStyle5 ? setChangeStyle5(false) : setChangeStyle5(true));
}

const changeLiStyle6 = () => {
    for(let i = 0; i < stateList.length; i++) {
        stateListFunc[i](false);
    }
    (changeStyle6 ? setChangeStyle6(false) : setChangeStyle6(true));
}

const changeLiStyle7 = () => {
    for(let i = 0; i < stateList.length; i++) {
        stateListFunc[i](false);
    }
    (changeStyle7 ? setChangeStyle7(false) : setChangeStyle7(true));
}

const changeLiStyle8 = () => {
    for(let i = 0; i < stateList.length; i++) {
        stateListFunc[i](false);
    }
    (changeStyle8 ? setChangeStyle8(false) : setChangeStyle8(true));
}

const changeLiStyle9 = () => {
    for(let i = 0; i < stateList.length; i++) {
        stateListFunc[i](false);
    }
    (changeStyle9 ? setChangeStyle9(false) : setChangeStyle9(true));
}

const changeLiStyle10 = () => {
    for(let i = 0; i < stateList.length; i++) {
        stateListFunc[i](false);
    }
    (changeStyle10 ? setChangeStyle10(false) : setChangeStyle10(true));
}


// changes state for center collumn and uses charge state to use in sup tag for the html presentation
const submitHandler = () => {
    for(let i = 0; i < stateList.length; i++) {
        if(stateList[i] === true) {
            setElectrode2(elements[i]);
            if(elements[i] === "Iron3") {
                setCharge2("3+");
                setElectrode2("Iron");
                console.log("Pause");
            } else if (elements[i] === "Iron2"){
```

```
            setCharge2("2+");
            setElectrode2("Iron");

        } else {
            setElectrode2(elements[i]);
            setCharge2("");
        }
    }
  }
}









  return (
    <div>
      <ul className='editElectrodeBox'>
        <li className={changeStyle1 ? 'electrodeBoxItem selected' : 'electrodeBoxItem'} onClick={changeLiStyle}
id='elementOne'>Magnesium</li>
        <li className={changeStyle2 ? 'electrodeBoxItem selected' : 'electrodeBoxItem'} onClick={changeLiStyle2}
id='2'>Strontium</li>
        <li className={changeStyle3 ? 'electrodeBoxItem selected' : 'electrodeBoxItem'} onClick={changeLiStyle3}
id='3'>Flurorine</li>
        <li className={changeStyle4 ? 'electrodeBoxItem selected' : 'electrodeBoxItem'} onClick={changeLiStyle4}
id='4'>Iron<sup>3+</sup></li>
        <li className={changeStyle5 ? 'electrodeBoxItem selected' : 'electrodeBoxItem'} onClick={changeLiStyle5}
id='5'>Hydrogen</li>
        <li className={changeStyle6 ? 'electrodeBoxItem selected' : 'electrodeBoxItem'}
onClick={changeLiStyle6}id='6'>Iron<sup>2+</sup></li>
        <li className={changeStyle7 ? 'electrodeBoxItem selected' : 'electrodeBoxItem'} onClick={changeLiStyle7}
id='7'>Sodium</li>
        <li className={changeStyle8 ? 'electrodeBoxItem selected' : 'electrodeBoxItem'} onClick={changeLiStyle8}
id='8'>Calcium</li>
        <li className={changeStyle9 ? 'electrodeBoxItem selected' : 'electrodeBoxItem'} onClick={changeLiStyle9}
id='9'>Lithium</li>
        <li className={changeStyle10 ? 'electrodeBoxItem selected' : 'electrodeBoxItem'} onClick={changeLiStyle10}
id='10'>Nickel</li>
      </ul>
      <button className='button-75' onClick={submitHandler}>Submit Electrode 2</button>
    </div>
  )
}

export default EditElectrodeTwo

//EditReactionConditions.jS
import React, { useState } from 'react'

const EditReactionConditions = ({setTemperature, setFlowRate, setPressure}) => {

  const[reactionError, setReactionError] = useState("");
  const[details, setDetails] = useState({temperature: "", flowRate: "", pressure: ""});



  const submitHandler = (e) => {
    e.preventDefault();
    if (/^[0-9]+$/.test(details.temperature) && /^[0-9]+$/.test(details.flowRate) && /^[0-9]+$/.test(details.pressure)) {
      setTemperature(details.temperature);
      setFlowRate(details.flowRate);
      setPressure(details.pressure);
      setReactionError("");
    } else {
```

```
        setReactionError("Please enter integer values only for all input fields");
      }

    }
    return (
      <div>
        <form className='signUpForm'>
        <div className='signUpFormUser form-group'>
          <label>Temperature(K)</label>
          <input className='form-control' type='text' placeholder='Enter Temperature' onChange={e => setDetails({...details,
temperature: e.target.value})} value={details.temperature}></input>
        </div>
        <div className='signUpPass form-group'>
          <label>Flow Rate(m<sup>3</sup>/s)</label>
          <input className='form-control' type='text' placeholder='Enter Flow Rate' onChange={e => setDetails({...details,
flowRate: e.target.value})} value={details.flowRate}></input>
        </div>
        <div className='signUpConfirm form-group'>
          <label>Pressure(Pa)</label>
          <input className='form-control' type='text' placeholder='Enter Pressure' onChange={e => setDetails({...details,
pressure: e.target.value})} value={details.pressure}></input>
        </div>
        <div className='buttonContainerTwo buttonStraightener'>
          <button type="submit" className="button-75 buttonsSign2" role="button" onClick={submitHandler}><span
class="text">Submit</span></button>
        </div>
        <div>{reactionError}</div>



      </form>

      </div>
    )
}

export default EditReactionConditions

//EditReagentOne.js
import React from 'react'
import { useState } from 'react';

const EditReagentOne = ({reagent1, setReagent1}) => {


    const [changeStyle1, setChangeStyle1] = useState(false);
    const [changeStyle2, setChangeStyle2] = useState(false);
    const [changeStyle3, setChangeStyle3] = useState(false);
    const [changeStyle4, setChangeStyle4] = useState(false);
    const [changeStyle5, setChangeStyle5] = useState(false);
    const [changeStyle6, setChangeStyle6] = useState(false);
    const [changeStyle7, setChangeStyle7] = useState(false);
    const [changeStyle8, setChangeStyle8] = useState(false);
    const [changeStyle9, setChangeStyle9] = useState(false);
    const [changeStyle10, setChangeStyle10] = useState(false);

    const elements =["Benzene", "Toluene", "Methylene", "Iron3", "Hydrogen", "Iron2", "Sodium", "Calcium", "Lithium", "Nickel"];

    const stateList = [changeStyle1, changeStyle2, changeStyle3, changeStyle4, changeStyle5, changeStyle6, changeStyle7,
changeStyle8, changeStyle9, changeStyle10];

    const stateListFunc = [setChangeStyle1, setChangeStyle2, setChangeStyle3, setChangeStyle4, setChangeStyle5,
setChangeStyle6, setChangeStyle7, setChangeStyle8, setChangeStyle9, setChangeStyle10];


    const changeLiStyle = () => {
```

```
    for(let i = 0; i < stateList.length; i++) {
      stateListFunc[i](false);
    }
    (changeStyle1 ? setChangeStyle1(false) : setChangeStyle1(true));
  }

  const changeLiStyle2 = () => {
    for(let i = 0; i < stateList.length; i++) {
      stateListFunc[i](false);
    }
    (changeStyle2 ? setChangeStyle2(false) : setChangeStyle2(true));
  }

  const changeLiStyle3 = () => {
    for(let i = 0; i < stateList.length; i++) {
      stateListFunc[i](false);
    }
    (changeStyle3 ? setChangeStyle3(false) : setChangeStyle3(true));
  }

  const changeLiStyle4 = () => {
    for(let i = 0; i < stateList.length; i++) {
      stateListFunc[i](false);
    }
    (changeStyle4 ? setChangeStyle4(false) : setChangeStyle4(true));
  }

  const changeLiStyle5 = () => {
    for(let i = 0; i < stateList.length; i++) {
      stateListFunc[i](false);
    }
    (changeStyle5 ? setChangeStyle5(false) : setChangeStyle5(true));
  }

  const changeLiStyle6 = () => {
    for(let i = 0; i < stateList.length; i++) {
      stateListFunc[i](false);
    }
    (changeStyle6 ? setChangeStyle6(false) : setChangeStyle6(true));
  }

  const changeLiStyle7 = () => {
    for(let i = 0; i < stateList.length; i++) {
      stateListFunc[i](false);
    }
    (changeStyle7 ? setChangeStyle7(false) : setChangeStyle7(true));
  }

  const changeLiStyle8 = () => {
    for(let i = 0; i < stateList.length; i++) {
      stateListFunc[i](false);
    }
    (changeStyle8 ? setChangeStyle8(false) : setChangeStyle8(true));
  }

  const changeLiStyle9 = () => {
    for(let i = 0; i < stateList.length; i++) {
      stateListFunc[i](false);
    }
    (changeStyle9 ? setChangeStyle9(false) : setChangeStyle9(true));
  }

  const changeLiStyle10 = () => {
    for(let i = 0; i < stateList.length; i++) {
      stateListFunc[i](false);
```

```
    }
    (changeStyle10 ? setChangeStyle10(false) : setChangeStyle10(true));
  }

  const submitHandler = () => {
    for(let i = 0; i < stateList.length; i++) {
      if(stateList[i] === true) {
        setReagent1(elements[i]);
      }
    }
  }

  return (
    <div>
      <ul className='editElectrodeBox'>
      <li className={changeStyle1 ? 'electrodeBoxItem selected' : 'electrodeBoxItem'} onClick={changeLiStyle}
id='elementOne'>Benzene</li>
        <li className={changeStyle2 ? 'electrodeBoxItem selected' : 'electrodeBoxItem'} onClick={changeLiStyle2}
id='2'>Toluene</li>
        <li className={changeStyle3 ? 'electrodeBoxItem selected' : 'electrodeBoxItem'} onClick={changeLiStyle3}
id='3'>Methylene</li>
        <li className={changeStyle4 ? 'electrodeBoxItem selected' : 'electrodeBoxItem'} onClick={changeLiStyle4}
id='4'>Iron<sup>3+</sup></li>
        <li className={changeStyle5 ? 'electrodeBoxItem selected' : 'electrodeBoxItem'} onClick={changeLiStyle5}
id='5'>Hydrogen</li>
        <li className={changeStyle6 ? 'electrodeBoxItem selected' : 'electrodeBoxItem'}
onClick={changeLiStyle6}id='6'>Iron<sup>2+</sup></li>
        <li className={changeStyle7 ? 'electrodeBoxItem selected' : 'electrodeBoxItem'} onClick={changeLiStyle7}
id='7'>Sodium</li>
        <li className={changeStyle8 ? 'electrodeBoxItem selected' : 'electrodeBoxItem'} onClick={changeLiStyle8}
id='8'>Calcium</li>
        <li className={changeStyle9 ? 'electrodeBoxItem selected' : 'electrodeBoxItem'} onClick={changeLiStyle9}
id='9'>Lithium</li>
        <li className={changeStyle10 ? 'electrodeBoxItem selected' : 'electrodeBoxItem'} onClick={changeLiStyle10}
id='10'>Nickel</li>
      </ul>
      <div className='electrode1submit'><button className='button-75' onClick={submitHandler}>Submit Reagent
One</button></div>
    </div>
  )
}

export default EditReagentOne
//EditReagentTwo.js
import React from 'react'
import { useState } from 'react';

const EditReagentTwo = ({reagent2, setReagent2}) => {


  const [changeStyle1, setChangeStyle1] = useState(false);
  const [changeStyle2, setChangeStyle2] = useState(false);
  const [changeStyle3, setChangeStyle3] = useState(false);
  const [changeStyle4, setChangeStyle4] = useState(false);
  const [changeStyle5, setChangeStyle5] = useState(false);
  const [changeStyle6, setChangeStyle6] = useState(false);
  const [changeStyle7, setChangeStyle7] = useState(false);
  const [changeStyle8, setChangeStyle8] = useState(false);
  const [changeStyle9, setChangeStyle9] = useState(false);
  const [changeStyle10, setChangeStyle10] = useState(false);

  const elements =["Benzene", "Toluene", "Methylene", "Iron3", "Hydrogen", "Iron2", "Sodium", "Calcium", "Lithium", "Nickel"];

  const stateList = [changeStyle1, changeStyle2, changeStyle3, changeStyle4, changeStyle5, changeStyle6, changeStyle7,
changeStyle8, changeStyle9, changeStyle10];
```

```
  const stateListFunc = [setChangeStyle1, setChangeStyle2, setChangeStyle3, setChangeStyle4, setChangeStyle5,
setChangeStyle6, setChangeStyle7, setChangeStyle8, setChangeStyle9, setChangeStyle10];


  const changeLiStyle = () => {
    for(let i = 0; i < stateList.length; i++) {
      stateListFunc[i](false);
    }
    (changeStyle1 ? setChangeStyle1(false) : setChangeStyle1(true));
  }

  const changeLiStyle2 = () => {
    for(let i = 0; i < stateList.length; i++) {
      stateListFunc[i](false);
    }
    (changeStyle2 ? setChangeStyle2(false) : setChangeStyle2(true));
  }

  const changeLiStyle3 = () => {
    for(let i = 0; i < stateList.length; i++) {
      stateListFunc[i](false);
    }
    (changeStyle3 ? setChangeStyle3(false) : setChangeStyle3(true));
  }

  const changeLiStyle4 = () => {
    for(let i = 0; i < stateList.length; i++) {
      stateListFunc[i](false);
    }
    (changeStyle4 ? setChangeStyle4(false) : setChangeStyle4(true));
  }

  const changeLiStyle5 = () => {
    for(let i = 0; i < stateList.length; i++) {
      stateListFunc[i](false);
    }
    (changeStyle5 ? setChangeStyle5(false) : setChangeStyle5(true));
  }

  const changeLiStyle6 = () => {
    for(let i = 0; i < stateList.length; i++) {
      stateListFunc[i](false);
    }
    (changeStyle6 ? setChangeStyle6(false) : setChangeStyle6(true));
  }

  const changeLiStyle7 = () => {
    for(let i = 0; i < stateList.length; i++) {
      stateListFunc[i](false);
    }
    (changeStyle7 ? setChangeStyle7(false) : setChangeStyle7(true));
  }

  const changeLiStyle8 = () => {
    for(let i = 0; i < stateList.length; i++) {
      stateListFunc[i](false);
    }
    (changeStyle8 ? setChangeStyle8(false) : setChangeStyle8(true));
  }

  const changeLiStyle9 = () => {
    for(let i = 0; i < stateList.length; i++) {
      stateListFunc[i](false);
    }
    (changeStyle9 ? setChangeStyle9(false) : setChangeStyle9(true));
```

```
    }

    const changeLiStyle10 = () => {
        for(let i = 0; i < stateList.length; i++) {
            stateListFunc[i](false);
        }
        (changeStyle10 ? setChangeStyle10(false) : setChangeStyle10(true));
    }

    // changes state for center collumn and uses charge state to use in sup tag for the html presentation

    const submitHandler = () => {
        for(let i = 0; i < stateList.length; i++) {
            if(stateList[i] === true) {
                setReagent2(elements[i]);
            }
        }
    }


    return (
        <div>
            <ul className='editElectrodeBox'>
            <li className={changeStyle1 ? 'electrodeBoxItem selected' : 'electrodeBoxItem'} onClick={changeLiStyle}
id='elementOne'>Benzene</li>
                <li className={changeStyle2 ? 'electrodeBoxItem selected' : 'electrodeBoxItem'} onClick={changeLiStyle2}
id='2'>Toluene</li>
                <li className={changeStyle3 ? 'electrodeBoxItem selected' : 'electrodeBoxItem'} onClick={changeLiStyle3}
id='3'>Methylene</li>
                <li className={changeStyle4 ? 'electrodeBoxItem selected' : 'electrodeBoxItem'} onClick={changeLiStyle4}
id='4'>Iron<sup>3+</sup></li>
                <li className={changeStyle5 ? 'electrodeBoxItem selected' : 'electrodeBoxItem'} onClick={changeLiStyle5}
id='5'>Hydrogen</li>
                <li className={changeStyle6 ? 'electrodeBoxItem selected' : 'electrodeBoxItem'}
onClick={changeLiStyle6}id='6'>Iron<sup>2+</sup></li>
                <li className={changeStyle7 ? 'electrodeBoxItem selected' : 'electrodeBoxItem'} onClick={changeLiStyle7}
id='7'>Sodium</li>
                <li className={changeStyle8 ? 'electrodeBoxItem selected' : 'electrodeBoxItem'} onClick={changeLiStyle8}
id='8'>Calcium</li>
                <li className={changeStyle9 ? 'electrodeBoxItem selected' : 'electrodeBoxItem'} onClick={changeLiStyle9}
id='9'>Lithium</li>
                <li className={changeStyle10 ? 'electrodeBoxItem selected' : 'electrodeBoxItem'} onClick={changeLiStyle10}
id='10'>Nickel</li>
            </ul>
            <button className='button-75' onClick={submitHandler}>Submit Reagent Two</button>
        </div>
    )
}

export default EditReagentTwo
//EditTubing.js

import React from 'react';
import { useState } from 'react';

const EditTubing = ({conditionsSetTube, setConditionsSetTube, tubeLength, setTubeLength, tubeDiameter, setTubeDiameter,
loops, setLoops, material, setMaterial}) => {

    const [details, setDetails] = useState({tubeLength: "", tubeDiameter: "", loops: "", material: ""});

    const [error, setError] = useState("");

    const submitHandler = (e) => {
        e.preventDefault();
        if (/^[0-9]+$/.test(details.tubeLength) && /^[0-9]+$/.test(details.tubeDiameter) && /^[0-9]+$/.test(details.loops)) {
            setConditionsSetTube(prevCount => prevCount + 1);
```

```
            setTubeLength([...tubeLength, details.tubeLength]);
            setTubeDiameter([...tubeDiameter, details.tubeDiameter]);
            setLoops([...loops, details.loops]);
            setMaterial([...material, details.material]);
            setError("");
        } else {
            setError("Please enter integer values for the first three fields.");
        }

    }
    return (
        <div>
            <form className='signUpForm'>
                <div className='signUpFormUser form-group'>
                    <label>Tube Length(mm)</label>
                    <input className='form-control' type='text' placeholder='Enter Tube Length' onChange={e =>
setDetails({...details, tubeLength: e.target.value})} value={details.tubeLength}></input>
                </div>
                <div className='signUpPass form-group'>
                    <label>Add Tube Diameter(mm)</label>
                    <input className='form-control' type='text' placeholder='Enter Tube Diameter' onChange={e =>
setDetails({...details, tubeDiameter: e.target.value})} value={details.tubeDiameter}></input>
                </div>
                <div className='signUpFormUser form-group'>
                    <label>Number of Loops</label>
                    <input className='form-control' type='text' placeholder='Enter Number of Loops' onChange={e =>
setDetails({...details, loops: e.target.value})} value={details.loops}></input>
                </div>
                <div className='signUpPass form-group'>
                    <label>Material</label>
                    <input className='form-control' type='text' placeholder='Enter Material' onChange={e => setDetails({...details,
material: e.target.value})} value={details.material}></input>
                </div>
                <div className='buttonContainerTwo'>
                    <button className="button-75 buttonsSign2 buttonStraightener2" onClick={submitHandler}><span
class="text">Submit</span></button>
                </div>
            </form>

            <div>{error}</div>


        </div>
    )
}

export default EditTubing

//EditVariableConditions.js

import React, { useState } from 'react'


const EditVariableConditions = ({temperatureV, setTemperatureV, temperatureVTime, setTemperatureVTime, pressureV,
setPressureV, pressureVTime, setPressureVTime, flowRateV, setFlowRateV, flowRateVTime, setFlowRateVTime,
conditionsSet, setConditionsSet, timeT, setTimeT, timeF, setTimeF, timeP, setTimeP}) => {

//these details states will keep track of the inputs and change the props states being passed through, timev1s = timeV1
(seconds). timev1st = timeV1 seconds in tens etc
const [detailsT, setDetailsT] = useState({temperatureV: "", timeV1s: "", timeV1m: "", timeV1h: "", timeV1: ""});
const [detailsP, setDetailsP] = useState({pressureV: "", timeV2s: "", timeV2m: "", timeV2h: "", timeV2: ""});
const [detailsF, setDetailsF] = useState({flowRateV: "", timeV3s: "", timeV3m: "", timeV3h: "", timeV3: ""});

const [error, setError] = useState("");
```

```
// submit handlers for all submits


let counter = 0;

const submitHandlerT = (e) => {
    e.preventDefault();
    if (/^[0-9]+$/.test(detailsT.temperatureV) && /^[0-9]+$/.test(detailsT.timeV1s) && /^[0-9]+$/.test(detailsT.timeV1m) && /^[0-9]+$/.test(detailsT.timeV1h)) {
        setTemperatureV([...temperatureV, detailsT.temperatureV]);

        //since we have 3 input fields for secs, mins, hours, we need to convert all to seconds then set timev to that state which will be saved in database
        //converting hours/mins to seconds and also parsing them as an int so that the total doesn't concat strings
        let timeTh2s = parseInt(detailsT.timeV1h * 3600);
        let timeTm2s = parseInt(detailsT.timeV1m * 60);
        let timeTs2s = parseInt(detailsT.timeV1s)
        let timeTTotal = timeTs2s + timeTh2s + timeTm2s;
        //before setting the tempVTime we need to convert all the fields and calculate it into seconds
        setTemperatureVTime([...temperatureVTime, timeTTotal]);
        setTimeT([...timeT, `${detailsT.timeV1h}h:${detailsT.timeV1m}m:${detailsT.timeV1s}s`]);
        //keeps a count of all the variabled conditions created
        setConditionsSet(prevCount => prevCount + 1);
        //clears error


        setError("");


    } else {
        setError("Please enter integer values for all fields")
    }


}

const submitHandlerP = e => {
    e.preventDefault();
    if (/^[0-9]+$/.test(detailsP.pressureV) && /^[0-9]+$/.test(detailsP.timeV2s) && /^[0-9]+$/.test(detailsP.timeV2m) && /^[0-9]+$/.test(detailsP.timeV2h)) {
        setPressureV([...pressureV, detailsP.pressureV]);
        let timePh2s = parseInt(detailsP.timeV2h * 3600);
        let timePm2s = parseInt(detailsP.timeV2m * 60);
        let timePs2s = parseInt(detailsP.timeV2s)
        let timePTotal = timePs2s + timePh2s + timePm2s;
        setPressureVTime([...pressureVTime, timePTotal])
        setTimeP([...timeP, `${detailsP.timeV2h}h:${detailsP.timeV2m}m:${detailsP.timeV2s}s`]);
        setConditionsSet(prevCount => prevCount + 1);
        setError("");
    } else {
        setError("Please enter integer values for all fields")
    }
}

const submitHandlerF = e => {
    e.preventDefault();
    if (/^[0-9]+$/.test(detailsF.flowRateV) && /^[0-9]+$/.test(detailsF.timeV3s) && /^[0-9]+$/.test(detailsF.timeV3m) && /^[0-9]+$/.test(detailsF.timeV3h)) {
        setFlowRateV([...flowRateV, detailsF.flowRateV]);
        let timeFh2s = parseInt(detailsF.timeV3h * 3600);
        let timeFm2s = parseInt(detailsF.timeV3m * 60);
        let timeFs2s = parseInt(detailsF.timeV3s)
        let timeFTotal = timeFs2s + timeFh2s + timeFm2s;
        setFlowRateVTime([...flowRateVTime, timeFTotal]);
        setTimeF([...timeF, `${detailsF.timeV3h}h:${detailsF.timeV3m}m:${detailsF.timeV3s}s`]);
        setConditionsSet(prevCount => prevCount + 1);
```

```
      setError("");
    } else {
      setError("Please enter integer values for all fields")
    }
}
  return (
    <div>
      <form className='signUpForm'>
        <div className='signUpFormUser form-group'>
          <label>Temperature(K)</label>
          <input className='form-control' type='text' placeholder='Enter Temperature' onChange={e =>
setDetailsT({...detailsT, temperatureV: e.target.value})} value={detailsT.temperatureV}></input>
        </div>
        <div className='hoursInputContainer'>
          <div className='hoursInputContainerCol1'>
            <label>Time(h)</label>
            <input className='form-control' type='text' placeholder='Hours' onChange={e => setDetailsT({...detailsT,
timeV1h: e.target.value})} value={detailsT.timeV1h}></input>
          </div>
          <div className="hoursInputContainerCol2">
            <label>Time(m)</label>
            <input className='form-control' type='text' placeholder='Minutes' onChange={e => setDetailsT({...detailsT,
timeV1m: e.target.value})} value={detailsT.timeV1m}></input>
          </div>
          <div className="hoursInputContainerCol3">
            <label>Time(s)</label>
            <input className='form-control' type='text' placeholder='Seconds' onChange={e => setDetailsT({...detailsT,
timeV1s: e.target.value})} value={detailsT.timeV1s}></input>
          </div>
        </div>
        <div className='buttonContainerTwo'>
          <button className="button-75 buttonsSign2" onClick={submitHandlerT}><span class="text"
id="onez">Submit</span></button>
        </div>
      </form>
      <form className='signUpForm'>
        <div className='signUpFormUser form-group'>
          <label>Pressure(Pa)</label>
          <input className='form-control' type='text' placeholder='Enter Pressure' onChange={e => setDetailsP({...detailsP,
pressureV: e.target.value})} value={detailsP.pressureV}></input>
        </div>
        <div className='hoursInputContainer'>
          <div className='hoursInputContainerCol1'>
            <label>Time(h)</label>
            <input className='form-control' type='text' placeholder='Hours' onChange={e => setDetailsP({...detailsP,
timeV2h: e.target.value})} value={detailsP.timeV2h}></input>
          </div>
          <div className="hoursInputContainerCol2">
            <label>Time(m)</label>
            <input className='form-control' type='text' placeholder='Minutes' onChange={e => setDetailsP({...detailsP,
timeV2m: e.target.value})} value={detailsP.timeV2m}></input>
          </div>
          <div className="hoursInputContainerCol3">
            <label>Time(s)</label>
            <input className='form-control' type='text' placeholder='Seconds' onChange={e => setDetailsP({...detailsP,
timeV2s: e.target.value})} value={detailsP.timeV2s}></input>
          </div>
        </div>
        <div className='buttonContainerTwo'>
          <button className="button-75 buttonsSign2" onClick={submitHandlerP}><span class="text">Submit</span>
</button>
        </div>
      </form>
      <form className='signUpForm'>
        <div className='signUpFormUser form-group'>
          <label>Flow Rate(m<sup>3</sup>/s)</label>
          <input className='form-control' type='text' placeholder='Enter Flow Rate' onChange={e =>
```

```
setDetailsF({...detailsF, flowRateV: e.target.value})} value={detailsF.flowRateV}></input>
            </div>
            <div className='hoursInputContainer'>
              <div className='hoursInputContainerCol1'>
                <label>Time(h)</label>
                <input className='form-control' type='text' placeholder='Hours' onChange={e => setDetailsF({...detailsF,
timeV3h: e.target.value})} value={detailsF.timeV3h}></input>
              </div>
              <div className="hoursInputContainerCol2">
                <label>Time(m)</label>
                <input className='form-control' type='text' placeholder='Minutes' onChange={e => setDetailsF({...detailsF,
timeV3m: e.target.value})} value={detailsF.timeV3m}></input>
              </div>
              <div className="hoursInputContainerCol3">
                <label>Time(s)</label>
                <input className='form-control' type='text' placeholder='Seconds' onChange={e => setDetailsF({...detailsF,
timeV3s: e.target.value})} value={detailsF.timeV3s}></input>
              </div>
            </div>
            <div className='buttonContainerTwo'>
              <button className="button-75 buttonsSign2" onClick={submitHandlerF}><span class="text">Submit</span>
</button>
            </div>
          </form>
          <div>{error}</div>

        </div>
    )
}

export default EditVariableConditions

//ShowSaveError.js
import React from 'react'

const ShowSaveError = ({saveErrorE1, saveErrorE2, saveErrorT, saveErrorF, saveErrorP, saveErrorR1, saveErrorR2,
saveErrorED, saveErrorEA}) => {







    return (
        <div>
          <div className='errorContainer'>
            <div className='errorRow1'>{saveErrorE1}</div>
            <div className='errorRow2'>{saveErrorE2}</div>
            <div className='errorRow3'>{saveErrorT}</div>
            <div className='errorRow4'>{saveErrorF}</div>
            <div className='errorRow5'>{saveErrorP}</div>
            <div className='errorRow6'>{saveErrorR1}</div>
            <div className='errorRow7'>{saveErrorR2}</div>
            <div className='errorRow8'>{saveErrorED}</div>
            <div className='errorRow9'>{saveErrorEA}</div>
            <div className='errorRow10'>{saveErrorE2}</div>

          </div>

        </div>
    )
}

export default ShowSaveError

//node OPC UA Client
```

```
import {
    OPCUAClient,
    MessageSecurityMode, SecurityPolicy,
    AttributeIds,
    makeBrowsePath,
    ClientSubscription,
    TimestampsToReturn,
    ClientMonitoredItem,
    DataValue
} from "node-opcua";




const connectionStrategy = {
    initialDelay: 1000,
    maxRetry: 1
}
const options = {
    applicationName: "MyClient",
    connectionStrategy: connectionStrategy,
    securityMode: MessageSecurityMode.None,
    securityPolicy: SecurityPolicy.None,
    endpointMustExist: false,
};
const client = OPCUAClient.create(options);
// const endpointUrl = "opc.tcp://opcuademo.sterfive.com:26543";
const endpointUrl = "opc.tcp://sebastians-mbp.lan:4334/UA/MyLittleServer";
async function main() {
  try {
    // step 1 : connect to
        await client.connect(endpointUrl);
        console.log("connected !");

    // step 2 : createSession
        const session = await client.createSession();
        console.log("session created !");




        const dataValue2 = await session.read({
            nodeId: "ns=1;s=temperature",
            attributeId: AttributeIds.Value});
        console.log(" value = " , dataValue2.toString());


    // close session
        await session.close();

    // disconnecting
        await client.disconnect();

   console.log("done");




  } catch(err) {
    console.log("An error has occured : ",err);
  }

}

main();
```

```
//initiate.sh (client directory)
node ../../sample_client_ts/sample_client.js

//Opcua sample server.js
/*global require,setInterval,console */
const opcua = require("node-opcua");

// Let's create an instance of OPCUAServer
const server = new opcua.OPCUAServer({
    port: 4334, // the port of the listening socket of the server
    resourcePath: "/UA/MyLittleServer", // this path will be added to the endpoint resource name
     buildInfo : {
        productName: "MySampleServer1",
        buildNumber: "7658",
        buildDate: new Date(2022,8,8)
    }
});

function post_initialize() {
    console.log("initialized");
    function construct_my_address_space(server) {

        const addressSpace = server.engine.addressSpace;
        const namespace = addressSpace.getOwnNamespace();

        // declare a new object
        const device = namespace.addObject({
            organizedBy: addressSpace.rootFolder.objects,
            browseName: "MyDevice"
        });

        //delcaration of MR object
        const microReactor = namespace.addObject({
            organizedBy: addressSpace.rootFolder.objects,
            browseName: "Microreactor"
        })

        //declaration of variables to be overwritten


        let temperature =20;//13;

        // temperature
        namespace.addVariable({
            componentOf: microReactor,
            nodeId: "ns=1;s=temperature",
            browseName: "temperature",
            dataType: "Double",
            value: {
                get: function () {
                    return new opcua.Variant({dataType: opcua.DataType.Double, value: temperature });
                }

            }
        })









        // add a variable named MyVariable1 to the newly created folder "MyDevice"
        let variable1 = 1;
```

```
    // emulate variable1 changing every 500 ms
    setInterval(function(){  variable1+=1; }, 500);



    namespace.addVariable({
        componentOf: device,
        nodeId: "ns=1;b=118118F",
        browseName: "MyVariable1",
        dataType: "Double",
        value: {
            get: function () {
                return new opcua.Variant({dataType: opcua.DataType.Double, value: variable1 });
            }
        }
    });



    // add a variable named MyVariable2 to the newly created folder "MyDevice"
    let variable2 = 10.0;

    namespace.addVariable({

        componentOf: device,

        nodeId: "ns=1;b=1020FFAA", // some opaque NodeId in namespace 4

        browseName: "MyVariable2",

        dataType: "Double",

        value: {
            get: function () {
                return new opcua.Variant({dataType: opcua.DataType.Double, value: variable2 });
            },
            set: function (variant) {
                variable2 = parseFloat(variant.value);
                return opcua.StatusCodes.Good;
            }
        }
    });
    const os = require("os");
    /**
     * returns the percentage of free memory on the running machine
     * @return {double}
     */
    function available_memory() {
        // var value = process.memoryUsage().heapUsed / 1000000;
        const percentageMemUsed = os.freemem() / os.totalmem() * 100.0;
        return percentageMemUsed;
    }
    namespace.addVariable({

        componentOf: device,

        nodeId: "s=free_memory", // a string nodeID
        browseName: "FreeMemory",
        dataType: "Double",
        value: {
            get: function () {return new opcua.Variant({dataType: opcua.DataType.Double, value: available_memory() });}
        }
    });
}
construct_my_address_space(server);
```

```
    server.start(function() {
        console.log("Server is now listening ... ( press CTRL+C to stop)");
        console.log("port ", server.endpoints[0].port);
        const endpointUrl = server.endpoints[0].endpointDescriptions()[0].endpointUrl;
        console.log(" the primary server endpoint url is ", endpointUrl );
    });
}
server.initialize(post_initialize);
//initiate.sh (server directory)
node ../../myserver/sample_server.js
```

```css
.App {
  height: 100vh;
  overflow: hidden !important;

}

.introBox {
  border: 2px solid black;
  position: absolute;
  top: 20%;
  left: 50%;
  transform: translate(-50%, 0);
  background-image: linear-gradient(to right, rgb(162, 234, 142) , rgb(118, 203, 240));

}


* {
  margin: 0;
  padding: 0;
  box-sizing: border-box;
}

.button-75 {
  align-items: center;
  background-image: linear-gradient(135deg, #f34079 40%, #fc894d);
  border: 0;
  border-radius: 10px;
  box-sizing: border-box;
  color: #fff;
  cursor: pointer;
  display: flex;
  flex-direction: column;
  font-family: "Codec cold",sans-serif;
  font-size: 16px;
  font-weight: 700;
  height: 54px;
  justify-content: center;
  letter-spacing: .4px;
  line-height: 1;
  max-width: 30%;
  min-width: 15%;
  padding-left: 30px;
  padding-right: 30px;
  padding-top: 3px;
  margin-left: 35%;
  text-decoration: none;
  text-transform: uppercase;
  user-select: none;
  -webkit-user-select: none;
  touch-action: manipulation;



}

.button-75:active {
  outline: 0;
}

.button-75:hover {
  outline: 0;
}
```

```css
.button-75 span {
  transition: all 200ms;
}

.button-75:hover span {
  transform: scale(.9);
  opacity: .75;
}

@media screen and (max-width: 991px) {
  .button-75 {
    font-size: 15px;
    height: 50px;
  }

  .button-75 span {
    line-height: 50px;
  }
}

.button-76 {
  align-items: center;
  background-image: linear-gradient(135deg, #f34079 40%, #fc894d);
  border: 0;
  border-radius: 10px;
  box-sizing: border-box;
  color: #fff;
  cursor: pointer;
  display: flex;
  flex-direction: column;
  font-family: "Codec cold",sans-serif;
  font-size: 14px;
  font-weight: 700;
  height: 54px;
  justify-content: center;
  letter-spacing: .4px;
  line-height: 1;
  max-width: 30%;
  min-width: 250px !important;
  padding-left: 20px;
  padding-right: 20px;
  padding-top: 3px;
  margin-left: 25%;
  text-decoration: none;
  text-transform: uppercase;
  user-select: none;
  -webkit-user-select: none;
  touch-action: manipulation;
  white-space: pre;


}

.button-76:active {
  outline: 0;
}

.button-76:hover {
  outline: 0;
}

.button-76 span {
  transition: all 200ms;
}
```

```css
.button-76:hover span {
  transform: scale(.9);
  opacity: .75;
}

@media screen and (max-width: 991px) {
  .button-76 {
    font-size: 15px;
    height: 50px;
  }

  .button-76 span {
    line-height: 50px;
  }
}

.titleMicroReactor {
  font-family: 'Anybody';
  font-size: xxx-large;
  padding-left: 50px;
  padding-right: 50px;
  padding-bottom: 30px;
  padding-top: 15px;
  font-weight: bold;
}
.buttonContainer {
  display: grid;
  grid-template-rows: 75px 50px 25px;
}

.introItem1 {

}

.introItem2 {

}



.introItem3 {
  margin-left: auto;
  margin-right: 15px;
  color: blue !important;
  text-decoration: underline !important;




}

.introItem3:hover {
  cursor: pointer;
}

.buttonContainerTwo {
  display: grid;
  grid-template-columns: 50% 50%;

}

.buttonsSign1 {
  transform: translate(77%, 0px);
}
```

```css
.buttonsSign2 {
  transform: translate(-77%, 0px);


}

.home {
  height: 100vh;
  width: auto;

  background-image: url(./components/reactor.png);
  background-size: contain;
  background-position-x: 400px;
  background-position-y: -50px;
  background-repeat: no-repeat;

}

.layout {

  font-family: 'Poppins', sans-serif;
  background-image: linear-gradient(to right, rgb(162, 234, 142) , rgb(118, 203, 240));
  overflow: hidden;
  height: 9vh;

}

.slider ul{
  width: 100%;
  transition: 1s ease;
  display: flex;
  justify-content: center;
  align-items: center;



}


.slider ul li {
  margin-block: 0.5rem;
  margin-inline: 3rem;
  list-style: none;
}

.slider ul li a {
  font-size: 18px;
  text-decoration: none;
  color: #fff;
  font-weight: bolder;

}



.menu-icon .menu {
  display: none;
  color: #fff;
  font-size: 60px;
  cursor: pointer;
}

.closed {
  display: flex;
```

```
  justify-content: flex-start;
  width: 100%;
  display: none;

}
.close {
  cursor: pointer;
  color: white;



}



.closed .close {
  font-size: 40px;
}

@media (max-width:900px) {

  .slider {

    position: fixed;
    min-width: 100%;
    height: 100vh;
    top: -100%;
    background-image: linear-gradient(to right, rgb(162, 234, 142) , rgb(118, 203, 240));
    transition: 1s ease;
  }


  .slider ul {
    display: flex;
    flex-direction: column;
    align-items: center;
    list-style: none;

  }

  .slider ul li {
    margin-block: 2.5rem;
  }

  .slider ul li a {
    font-size: 25px;
    text-decoration: none;
    color: #fff;
  }

  .slider.active {
    top: 0;
    transition: 1s ease;

  }

  .menu-icon .menu {
    display: block;
    color: #fff;
    font-size: 50px;
    cursor: pointer;
  }

  .closed {
    display: flex;
```

```
    justify-content: flex-start;
    width: 100%;

  }
  .close {
    cursor: pointer;
    color: white;



  }

  .closed .close {
    font-size: 40px;
  }



}

.error {
  border: 2px solid black;
  border-radius: 5px;
  font-size: larger;
  text-align: center;
  width: 30%;
  display: flex;
  justify-content: center;
  position: absolute;
  left: 50%;
  transform: translate(-50%, 200%);
  background-color: rgb(250, 179, 179);
  color: red;


}

.successMessage {
  border: 2px solid black;
  border-radius: 5px;
  font-size: larger;
  text-align: center;
  width: 30%;
  display: flex;
  justify-content: center;
  position: absolute;
  left: 50%;
  transform: translate(-50%, 200%);
  background-color: rgb(179, 242, 177);
  color: rgb(4, 85, 9);
}

.createContainer {
  display: grid;
  height: 100vh;
  grid-template-columns: 33.3vh, 33.3vh, 33.3vh;
  grid-template-rows: 91vh;
  grid-template-areas:
    "createCol1 createCol2 createCol3";


}

.createCol1 {
  text-align: center;
```

```
  display: grid;
  grid-template-columns: 1fr;
  grid-template-rows: 1fr 1fr 1fr 1fr 1fr 1fr;
  grid-template-areas:
    "createButton1"
    "createButton2"
    "createButton3"
    "createButton4"
    "createButton5"
    "createButton6";
  padding-top: 25px;
  padding-bottom: 25px;
  border: #f34079 2px solid;
}

.createCol2 {
  text-align: center;
  border: #f34079 2px solid;
}

.createCol3 {
  text-align: center;
  border: #f34079 2px solid;
}

@media only screen and (max-width:992px) {
  .createContainer{
    grid-template-columns: 50vh 50vh;
    grid-template-rows: 100vh 100vh;
    grid-template-areas:
      "createCol1 createCol2"
      "createCol3 createCol3";
  }

  .App {
    height: 100vh;
    overflow: auto !important;

  }

}

@media only screen and (max-width:650px) {
  .createContainer{
    grid-template-columns: 100vh;
    grid-template-rows: 100vh 100vh 100vh;
    grid-template-areas:
      "createCol1"
      "createCol2"
      "createCol3";
  }

  .button-75 {

  }

}

.editElectrodeBox {
  height: 35vh;
  list-style: none;
  overflow-y: scroll;
  background-image: linear-gradient(to left, rgb(162, 234, 142) , rgb(118, 203, 240));
  padding: 0px;
```

```
}


body {
  margin: 0;
  font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', 'Roboto', 'Oxygen',
    'Ubuntu', 'Cantarell', 'Fira Sans', 'Droid Sans', 'Helvetica Neue',
    sans-serif;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
}

code {
  font-family: source-code-pro, Menlo, Monaco, Consolas, 'Courier New',
    monospace;
}


.electrodeBoxItem {
  padding: 20px;
  cursor: pointer;

}




.selected {
  background-image: linear-gradient(135deg, #f34079 40%, #fc894d);
  color: whitesmoke;
}

::-webkit-scrollbar {
  -webkit-appearance: none;
  width: 10px;
}

::-webkit-scrollbar-thumb {
  border-radius: 5px;
  background-color: rgba(0,0,0,.5);
  -webkit-box-shadow: 0 0 1px rgba(255,255,255,.5);

}

.viewContainer {
  display: grid;
  width: 100vh;
  height: auto;
  grid-template-columns: 100vh;
  grid-template-rows: 30vh 45vh;
  grid-template-areas:
    "viewCol1"
    "viewCol2";

}

.viewCol1 {
  border: #f34079 solid 2px;
  width: 210vh;
  height: 100vh;

}
```

```css
.viewCol2 {
  border: #f34079 solid 2px;
  width: 210vh;
  height: 100vh;
}

.viewContainer2 {
  display: grid;
  grid-template-columns: 68vh 68vh 68vh;
  grid-template-rows: 100vh;
}

.view2Col1 {
  border: #f34079 solid 2px;
}

.view2Col2 {
  border: #f34079 solid 2px;
}

.view2Col3 {
  border: #f34079 solid 2px;
}

.errorContainer {
  display: grid;
}

.hoursInputContainer {
  display: grid;
  grid-template-columns: 15vh 15vh 15vh;
  grid-template-rows: 10vh;
  grid-template-areas:
    "hoursInputContainerCol1" "hoursInputContainerCol2" "hoursInputContainerCol3";

}

.hoverMR {
  cursor: pointer;
  font-size: large;
  text-decoration: none;
  color: rgb(0, 0, 0);
  font-weight: bolder;


}

.viewColumn {
  display: grid;
  grid-template-columns: 140vh 15vh 15vh;
  grid-template-rows: 5vh;
  grid-template-areas:
    "viewCol10" "viewCol20" "viewCol30";
    background-image: linear-gradient(to right, rgb(162, 234, 142) , rgb(118, 203, 240));
    border: #f34079 solid 2px;
}


.button-77 {
  align-items: center;
  background-image: linear-gradient(135deg, #f34079 40%, #fc894d);
  border: 0;
  border-radius: 10px;
  box-sizing: border-box;
  color: #fff;
```

```
  cursor: pointer;
  display: flex;
  flex-direction: column;
  font-family: "Codec cold",sans-serif;
  font-size: 14px;
  font-weight: 700;
  height: 100%;
  justify-content: center;
  letter-spacing: .4px;
  line-height: 1;
  max-width: 30%;
  min-width: 100px !important;
  padding-left: 20px;
  padding-right: 20px;
  padding-top: 3px;
  margin-left: 25%;
  text-decoration: none;
  text-transform: uppercase;
  user-select: none;
  -webkit-user-select: none;
  touch-action: manipulation;
  white-space: pre;


}

.button-77:active {
  outline: 0;
}

.button-77:hover {
  outline: 0;
}

.button-77 span {
  transition: all 200ms;
}

.button-77:hover span {
  transform: scale(.9);
  opacity: .75;
}

.runButtonAdjust {
  margin-top: 10%;
}

.savedMr {

  font-size: large;
  text-decoration: none;
  color: rgb(0, 0, 0);
  font-weight: bolder;
}

.electrode1submit {
  padding-bottom: 12px;
}

.buttonStraightener {
  margin-left: 45%;
}

.buttonStraightener2 {
```

```
  margin-left: 100%;
}

.currentConds {
  padding-bottom: 10%;
}

.rc2container {
  display: grid;
  text-align: center;
  grid-template-columns: 1fr;
  grid-template-rows: 0.55fr 0.52fr 0.52fr 0.52fr 0.52fr 0.52fr 0.52fr 0.52fr 0.52fr 0.52fr 0.7fr 0.7fr;
  grid-template-areas:
    "rc21"
    "rc22"
    "rc23"
    "rc24"
    "rc25"
    "rc26"
    "rc27"
    "rc28"
    "rc29"
    "rc30"
    "rc31"
    "rc32";




}

.homeText {
  font-size: x-large;

}



.home1 {
  padding: 90px;
  transform: translate(-210px, 0px);
  display: grid;
  text-align: center;

  grid-template-columns: 1fr;
  grid-template-rows: 1fr 40px 90px;
  grid-template-areas:
    "home2"
    "home3"
    "home4"

  ;


}

.home2 {
  padding: 30px;

}

.home3 {
  padding: 200px;
  transform: translate(7%, 0);
}
```

```css
.home4 {
  padding: 190px;
}



#homeh2 {
  font-family: Georgia, 'Times New Roman', Times, serif;
  font-size: 50px !important;


}
```

```
//mongoDB Scheme
const mongoose = require("mongoose");

const MicroreactorSchema = new mongoose.Schema({

    user: {
        type: String
    },

    name: {
        type: Number
    },

    electrodeOne: {
        type: String
    },

    electrodeTwo: {
        type: String
    },

    temperature: {
        type: String
    },

    pressure: {
        type: String
    },

    flowRate: {
        type: String
    },

    reagentOne: {
        type: String
    },

    reagentTwo: {
        type: String

    },

    electrodeArea: {
        type: String
    },

    electrodeDistance: {
        type: String
    },

    vcTemp: [
        String,
    ],

    vcTempTime: [
        String
    ],

    vcPressure: [
        String,
    ],

    vcPressureTime: [
        String,
    ],

    vcFlowRate: [
```

```
        String,
    ],

    vcFlowRateTime: [
        String
    ],

    tubing: [
        String
    ],

    run: false

});

const MicroreactorsModel = mongoose.model("microreactors", MicroreactorSchema);

module.exports = MicroreactorsModel;

//expressJS server

const express = require("express");
const app = express();
const mongoose = require("mongoose");
const MicroreactorsModel = require('./models/microreactors');
const fs = require("fs");
const exec = require("child_process").exec;

const cors = require("cors");
const { stdout, stderr } = require("process");
const res = require("express/lib/response");

mongoose.connect("mongodb+srv://sebzxp:sebzxp@cluster0.vnar4r6.mongodb.net/?retryWrites=true&w=majority");

app.use(express.json());
app.use(cors());



const writeOpcuaServerT = (temp) => {

    fs.readFile('../../myserver/sample_server.js', 'utf-8', function (error, contents) {
        if (error) {
            console.log(err);
            return;
        }


    const replacedT = contents.replace(/let temperature =/g, `let temperature =${temp};//`);



    fs.writeFile('../../myserver/sample_server.js', replacedT, 'utf-8', function (err) {
        if (err) {
            console.log(err);
        }

        });



    });

}
```

```
//test opcua connection
testOpcuaClient = () => {

    exec('sh ../../sample_client_ts/initiate.sh', (error, stdout, stderr) => {

        if(error) {
            console.log(stderr);


            throw error;

        }
        console.log('stdout', stdout)

        return true;
    })
}




//run the file

runOpcuaClient = () => {
    exec('sh ../../sample_client_ts/initiate.sh', (error, stdout, stderr) => {
        if(error) {
            console.log(stderr);
            throw error;
        }
        console.log('stdout', stdout)
    })
}



//run the server
runOpcuaServer = () => {
    exec('sh ../../myServer/initiate.sh', (error, stdout, stderr) => {
        if(error) {
            console.log(stderr);
            throw error;
        }
        console.log('stdout', stdout)
    })
}



//read the client file
readOpcuaClient = () => {
    fs.readFile('../../sample_client_ts/sample_client.js', 'utf-8', (err, data) => {
        if(err) {
            console.log(err);
            return;
        }
        console.log(data);
    })

}




//write the client file
writeOpcuaClient = () => {
```

```
    fs.appendFile('../../sample_client_ts/sample_client.js', content, err => {
        if(err) {
            console.log(err)
            return
        }
        console.log("opcua client updated")

    })
}



app.get("/readOpcua", (req, res) => {
    readOpcuaClient();


});

app.get("/runOpcua", (req, res) => {
    runOpcuaClient();

});

app.get("/writeOpcua", (req, res) => {
    writeOpcuaClient();

});

app.get("/writeOpcuaServerRun", (req, res) => {
    exec('sh ../../myserver/initiate.sh', (error, stdout, stderr) => {
        if(error) {
            console.log(stderr);
            res.send(`Not connected error: ${error}`);
            throw error;
        }
        console.log('stdout + successful run', stdout)
        return;
    })
});

app.post("/writeOpcuaServer", async(req, res) => {

    console.log(res);
    temperatureString = String(temperature);
    console.log(temperatureString + "temp string");
    slicedTemperatureString = temperatureString.slice(16, -2);
    console.log(slicedTemperatureString);
    writeOpcuaServerT(slicedTemperatureString);
})

app.post("/postMicroreactor", async (req, res) => {
    const mr = req.body;
    const newMr = new MicroreactorsModel(mr);

    //updating the server file
    writeOpcuaServerT(mr.temperature);
    await newMr.save();
    //running the server file through child process
    runOpcuaServer()
    res.json(mr);




});
```

```
app.get("/testOpcua", (req, res) => {




      exec('sh ../../sample_client_ts/initiate.sh', (error, stdout, stderr) => {
         if(error) {
            console.log(stderr);
            res.send(`Not connected error: ${error}`);
            throw error;
         }

         if(stdout.substring(0,8) === "An error") {
            res.send("Error, please check terminal for details");
            console.log(stdout)
         } else {
            res.send(`Connection successful.`); }
            console.log(stdout)
         return;
      })




});






app.get("/getMicroreactors", (req, res) => {
   MicroreactorsModel.find({}, (err, result) => {
      if(err) {
         res.json(err);
      } else {
         res.json(result);
      }
   });



});



app.delete('/delete/:id', async (req, res) => {
   const id = req.params.id
   await MicroreactorsModel.findByIdAndRemove(id).exec();
   res.send("Item Deleted");
});




app.listen(3001, () => {
   console.log("server running");
});
```