12-2023

# Developing a Flexible System for a Friendly Robot to Ease Dementia (FRED) Using Cloud Technologies and Software Design Patterns

Robert James Bray
*University of Tennessee, Knoxville*, rbray2@vols.utk.edu

To the Graduate Council:

I am submitting herewith a thesis written by Robert James Bray entitled "Developing a Flexible System for a Friendly Robot to Ease Dementia (FRED) Using Cloud Technologies and Software Design Patterns." I have examined the final electronic copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Computer Science.

Jens Gregor, Major Professor

We have read this thesis and recommend its acceptance:

Jens Gregor, Xiaopeng Zhao, James Plank

Accepted for the Council:
<u>Dixie L. Thompson</u>

Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

To the Graduate Council:

I am submitting herewith a thesis written by Robert James Bray entitled "Developing a Flexible System for a Friendly Robot to Ease Dementia (FRED) Using Cloud Technologies and Software Design Patterns." I have examined the final paper copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Computer Science.

Dr. Jens Gregor, Major Professor

We have read this thesis
and recommend its acceptance:

Dr. Jens Gregor

Dr. Xiaopeng Zhao

Dr. James Plank

Accepted for the Council:

Dixie L. Thompson

Vice Provost and Dean of the Graduate School

To the Graduate Council:
I am submitting herewith a thesis written by Robert James Bray entitled "Developing a Flexible System for a Friendly Robot to Ease Dementia (FRED) Using Cloud Technologies and Software Design Patterns." I have examined the final electronic copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Computer Science.

Dr. Jens Gregor, Major Professor

We have read this thesis
and recommend its acceptance:

Dr. Jens Gregor

Dr. Xiaopeng Zhao

Dr. James Plank

Accepted for the Council:

Dixie L. Thompson

Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

# Developing a Flexible System for a Friendly Robot to Ease Dementia (FRED) Using Cloud Technologies and Software Design Patterns

A Thesis Presented for

The Master of Science

Degree

The University of Tennessee, Knoxville

Robert James Bray

December 2023

*I would like to dedicate this work to my family: Haley, Micah, Mom, Dad, and Cashew.*

# Abstract

In this work, we designed two prototypes for a friendly robot to ease dementia (FRED). This affordable social robot is designed to provide company to older adults with cognitive decline, create reminders for important events and tasks, like taking medication, and providing cognitive stimulus through games. This project combines several cloud technologies including speech-to-text, cloud data storage, and chat generation in order to provide high level interactions with a social robot. Software design patterns were employed in the creation of the software to produce flexible code base that can sustain platform changes easily, including the framework used for the graphical user interface (GUI) and the database platform being used to store user data. The first prototype was developed on an Android-based system with an Arduino. This system was found to be expensive, unreliable, and difficult to develop on. The second prototype was therefore developed for a Raspberry Pi programmed using Python. Multiple tests with potential users were conducted in order to assess the capabilities and usability of the software created. These user tests showed overall satisfaction with the usability, and provided useful feedback for improving the software and expanding the capabilities of FRED.

# Table of Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

## 1.1 Problem Statement

Around the world, 10 million new cases of people living with dementia are reported every year [1]. In the United States alone, about 6.5 million people age 65 live with Alzheimer's Disease (AD) or related dementias (ADRD) [1]. By the year 2025, that number is predicted to grow to 7.1 million, and by 2050, 12.7 million [1]. As a result of this, the number of paid caregivers needs to increase by 235% to meet this rapid growth in the aging population [2]. However, caregiving is a stressful occupation that causes significant stress and long-term negative effects on mental health [3]. As a result, new solutions are required to meet these growing needs.

With the rapid growth of artifical intelligence (AI) and robotics technology, socially assistive robots (SARs) are developing into a feasible solution for people living with ADRD. Using AI, these SARs are able to assist caregivers in delivering better care, increase their quality of life, and improve the mental health of both people living with dementia (PLWDs) and their caregivers [4]–[6]. SARs with human-like qualities, like Pepper and NAO [7], [8], have been shown to be effective for PLWDs in providing therapy for their cognitive abilities, communication, and motor skills [9]. Additionally, robotic animals, like PARO and aibo [10], [11], are capable of positively reducing stress and loneliness in PLWDs [12]. Although these solutions have been proven to have positive impacts on PLWDs, they are well-known for their high price tags. For example, NAO can cost up to $10,000. This price tag can be extremely prohibitive for families, especially if they are already paying for professional care for their loved one. In this work, a friendly robot for easing dementia (FRED) was developed with the goal of solving this disparity in affordable SARS.

## 1.2 Current Market Solutions

Currently, several robots have been marketed having the same purpose as FRED. PARO is a large white seal robot which is able to detect different senses, like touch and hearing, and respond to its users through movement and comforting sounds, according to the creator's website [10]. PARO has the ability to remember previous actions that made the user happy, which it measures by how many times the user pets it. The goal is to recreate the success of animal therapy with patients by creating a robot that is a soft, cute white seal that makes comforting noises and responds to the user's touch. In a systematic review of PARO studies, Kang et al. found that PARO was capable of decreasing anxiety and agitation in users with dementia, as well as increasing quality of life, as measured by the QOL-Alzheimer's Disease scale and the amount of social interaction that they engage in after using PARO [13]. Another important result reported by the review was the reduction in psychotropic medications. Additionally, it is worth noting that the pulse rates in users were significantly lower after interacting with the robot. Lastly, there were no significant changes in cognition in users with dementia reported by any of the studies reviewed in this work. It is apparent that PARO significantly impacts users' well-being and mood, however, this improvement comes with a $6,000 price tag.

Another robot called ElliQ features a tablet for touch interaction, a humanoid appearance with glowing LED's, and a humanoid voice to interact with users. ElliQ keeps reminders for its users about medication and appointments, keeps users in contact with friends and family, and provides entertainment through games and exercise videos. In a study done by Chang et al. regarding the acceptance of voices used for robotic assistants, a short video of ElliQ is shown to participants in order to demonstrate what a robotic assistant might look and sound like [14]. No conclusions were drawn about what the effects of ElliQ may be on participants [15]. The results of this study reflect the general academic research regarding ElliQ. There has not been much research performed on the system, and the research that has been completed is inconclusive, whereas the effects of the PARO robot are well-researched.

The goal of designing the FRED robot is to aid PLWD and caregivers, while maintaining a price under $300. FRED must to be an effective solution that PLWDs can use everyday to give them company and entertainment, while caregivers have the ability to create helpful reminders for medication and events, as well as keep the PLWD in touch with their loved ones. FRED is a fully 3D-printed robot based on affordable computational resources.

## 1.3 Software Design Principles

In order to create a lasting system for conducting research and a viable product, careful consideration of the design process and principles need to be taken in order to make the long-term development of this system achievable and sustainable. Since this project is being designed from the ground up, the libraries and APIs being used are subject to change in the future. Thus, the design of the software architecture is critical. In order to accomplish this, background on proper software design practice is necessary. First, a look at the basic design principles of quality software architecture. According to Robert C. Martin, there are four basic ”symptoms” of the degradation of software: fragility, rigidity, immobility, and viscosity [16]. Fragility describes the weak state of a software architecture in the face of changes to modules or requirements. Rigidity is the unwillingness to change or refactor software that has a problem but would be extremely time consuming to fix. Immobility is the inability to reuse pieces of existing software in other projects due to its dependencies on other software. Lastly, viscosity describes two factors. When a change needs to be made to software, there are “design preserving” methods maintain the current design of the software, and there are ”hacks” which are quick fixes that may not maintain the existing software design. When the “design preserving” method is slower than the “hack,” Martin says that this is a “high viscosity” project. This makes intuitive sense to any software engineer, as there are often both “good” and “hacky” ways to solve a problem, and engineers often want to employ the ”good” methods for better software maintenance in the future. These four problems are important to consider when designing software, as they may determine its lifespan. Objected-oriented design and design patterns are two methods of mitigating these issues in software development.

In object-oriented design, it is important to create a base for the program that can be extended when new features are needed. According to Bertrand Meyer, modules should follow the Open-Closed Principle, which says that a module should be both open and closed, as the name suggests. An open module can be extended to contain more features that are required by an expanding software. A module that is closed can be used by other parts of the software with a strict-enough definition that its fundamental components will not be changed [17]. Meyer gives an extensive example of an abstract module A and its dependencies, but it is also possible to discover why this is important by thinking of a simple module like a Reminder, which will hold information for a reminder to some event, like medication. A Reminder instance would hold the time and title of the future event. It could be possible that another module would like to expand upon this by adding a summary of the event and an end-time for things like a doctor’s appointment or a concert. Another module may want to add a location, guests, etc. Adding to the original module will create a refactoring disarray in other dependencies, where the Reminder now needs many pieces of information that are not needed by other modules. As Meyer illustrates, inheritance is an easy way to solve this. The Reminder class could have a child class called Event, which will have a location, end-time, and summary information. This will allow new modules to use the added features without creating problems in previous dependencies. This principle will be followed in the design patterns work, which will be discussed in Chapter 3.

Design patterns are another important component of modular design. Gamma et al. were the first to define a design pattern in the context of computer science, where previously these design patterns were understood but not formally defined [18]. According to Gamma et al., a design pattern is a way to "identify, name, and abstract common themes in object-oriented design." These patterns describe abstract solutions to common problems in software design without alluding to specific implementation. They are a way for designers and programmers to communicate about a well-defined, well-tested solution. The most important aspect of design patterns is that they are abstract. They do not give instructions for implementing the solution to a problem, rather they give the framework for tackling this problem. Gamma et al. gives three different ways to describe what problem a design pattern is solving: creational, structural, and behavioral. Creational patterns solve problems regarding the creation of objects. Structural patterns describe the inheritance structure of a program. Lastly, behavioral patterns define the way in which multiple objects should interact with each other. These three "characterizations" are the fundamental problems that design patterns tackle. One design pattern that is considerably useful for a project like FRED is the Model-View-Controller (MVC) pattern, which is defined by Krasner et al. in the context of the Smalltalk-80 system [19]. Smalltalk is a development environment designed for educating programmers. Krasner et al. defined the MVC pattern as a way to separate the "interactive application components" in a modular way. The model handles the information of the program, the view handles the graphical components of showing the user the program, and the controller handles events from input devices and other logic in the program. This fundamental definition describes a coupled system, in which the view understands what the model contains. A variation of this pattern will be used in order to create a less-coupled system. Another design pattern that is useful for this project is the Data Mapper pattern, which is a way to abstract away the details of reading and writing information to a database. This pattern was named by Martin Fowler [20]. When database code is written throughout the code base, a change in database platform will be a costly change, because all of these instances of database code need to be changed. This Data Mapper class creates a layer of abstraction that mitigates this ripple-effect. The data mappers in this work will be simple mappers that support CRUD operations. Although this implementation is simple, it lays the foundation for future complexities that will arise as FRED becomes more capable. One last pattern that is helpful for this work is the Observer-Subject pattern. It seems that this pattern is not attributed to one person, rather a well-known and understood pattern throughout software engineering. The observer-subject pattern is often used in event-based applications, which is clearly applicable to this project. It creates much more succinct integration of events within the code base, since it provides abstract functions for any class to implement its own way of handling particular events that arise in the Subject.

## 1.4 Generative Language Models

One major aspect of social robotics is the interaction between a human user and a robotic agent. It is important that the interactions produced by the robotic agent feel natural and human-like, otherwise the user may feel like they are just interacting with nothing more than a computer. Generative models have made great progress in the last few years, especially with the evolution of OpenAI's GPT models. These models are changing the face of natural language processing (NLP).

How do these large language models (LLM) work? On a basic level, these models are based on the transformer architecture. Originally introduced by Vaswani et al., the transformer architecture is an encoder-decoder architecture which takes a sequence of tokens as input and produces a sequence of tokens as output [21]. Vaswani et al. were the first to apply the attention concept to this new transformer model, whereas previously it was used in recurrent neural networks (RNN). Their attention mechanism focuses on their "scaled dot-product attention," which is the dot product of the query and key vectors divided by $\sqrt{d_k}$, where $d_k$ is the dimension of the key vector. The scaling factor helps mitigate the effect of exploding gradients, since these matrices are being multiplied together. Additionally, the model laid about by Vaswani and co-researchers uses their multi-head attention mechanism, which enables the model to focus on different aspects of the input. The input is split into the same number of subsequences as there are attention heads. After processing these inputs, the outputs of the attention heads are concatenated together before they are fed into the next layer. Additionally, this feature enables easy parallelization of the attention task. This transformer model has been revolutionary in the NLP space, as previous models have struggled with context in long sequences of text. Now, with its attention innovations, the model is able to keep long-lasting context for the sequence and create coherent completions.

To talk more specifically about the GPT models, they are decoder-only transformer architectures, which means that they do not have a dedicated encoder component to their structure. Encoder-decoder architectures are traditionally used in fine-tuned tasks like automatic machine translation from one language to another. According to Brown et al., the goal of the GPT models is to be able to train a model that can better generalize on a broad range of tasks than fine-tuned models [22]. This work by Brown et al. is on the GPT-3 model, which performs much better than the previous GPT-2 model. This improvement was gained by greatly increasing the model's parameters to 175 billion, slightly modifying the model architecture, and training on "few-shot," "one-shot," and "zero-shot" examples. The modification in the model architecture is the addition of alternating dense and sparse attention layers. The training of the model is a significant contribution to training in the NLP space, as it is a new way of creating a more generalized model. Previously, language models were trained on large datasets in order to achieve good performance on a specific task. This is called fine-tuning. GPT-3 was not trained this way, rather it was trained with subsets of examples. In few-shot learning, the model is given a task description and a number of examples of input and output on a task, and its weights are not changed during this portion. Then, it is given one final input and is expected to give the output. This is the step where its weights will change. According to Brown et al., the goal is to train the model to give it a broader distribution than fine-tuned models. One-shot learning is the same as few-shot, but only one example is given. In zero-shot learning, the model is given only the task description and the input. These researchers trained the model on their filtered version of the Common Crawl dataset. Brown et al. found that the GPT-3 model could perform close to the state-of-the-art, fine-tuned models for many tasks, although it still struggles with some tasks, like text synthesis.

Although GPT-3 was initially released in 2020, its successors GPT-3.5 and GPT-4 boast significant performance increases in language tasks. Although OpenAI has not released formal performance comparisons between GPT-3 and these new models, GPT-4 has 19% better performance over 3.5 according to OpenAI's evaluations [23]. Unfortunately, this technical report does not give many details about GPT-4's architecture due to safety and competitive concerns. The only detail given is that the model was trained using Reinforcement Learning from Human Feedback (RLHF), which is a method developed by Christiano et al. in which a reward function is developed with the help of human-created feedback on the system's actions [24]. Although these models perform much better than their predecessors, they are also actively collecting data from completion requests to further train themselves [25]. This raises data privacy concerns, especially in a vulnerable group like older adults living with ADRD. Until recently, users did not have the capability to opt out of this data collection. In the FRED focus group with PLWDs and their caregivers, participants unanimously agreed that they were okay with using AI, but only if their data was not being collected or used by these companies. More details about this focus group will be provided in Chapter 4. This data collection is concerning in an application like FRED, since the robot is designed to make users feel comfortable talking about how they feel, but collecting that data is a clear violation of the user's privacy. Thus, at this time, GPT-3 is the model of choice for this project, and a future upgrade to a better performing model will be considered with new developments in the LLM space regarding data privacy.

## 1.5 Summary of Work

This thesis documents the development of an affordable social robot system for older adults with ADRD and the use of software design principles and patterns in order to create a long-lasting system for development and research. This process includes prototyping multiple versions of the system and testing with real potential users. Additionally, multiple user-centered studies were performed in order to collect thoughts on the usability of the system, and to test the robustness of the system's capabilities. Dr. Xiaopeng Zhao, head of the University of Tennessee Knoxville's (UTK) Social Machines and AI Robotics Technology (SMART) lab, is the principle investigator of this work. Zhao's lab is focused on the application of AI and application development to social robotics as it pertains to people with ADRD and other cognitive decline. Dr. Kimberly Mitchell, Assistant Professor of the UTK School of Design, is the Co-Principle Investigator of the FRED project. Mitchell is the design lead for the project, guiding and directing the design of the 3D-printed body of FRED, as well as the design of the graphical user interface (GUI). Luke MacDougal, from the Mechanical, Aerospace, and Biomedical (MABE) department at UTK, is the lead mechanical engineer of the project, managing all aspects of 3D-printing, including creating all 3D prints and advising on effective ways to produce a design concept through 3D-printing. Ella Hosse and Matthew Rightsell, from the UTK School of Design, are the designers that invented the screen designs for the GUI, created inspiration and designs for the 3D-printed bodies, as well as any other design work for the FRED project.

# Chapter 2

# Android Phone and Arduino

This chapter contains previously published work [26]. I was the primary author of this work. Dr. Xiaopeng Zhao and Dr. Kimberly Mitchell were the primary investigators. Luke MacDougall and Cody Blankenship were major contributors to the development of the FRED prototypes described here. Fengpei Yuan helped write the manuscript and perform labeling on the experiment videos from the usability study. Sylvia Cerel-Suhl proofread and contributed edits to the manuscript. Reproduced with permission from Springer Nature.

## 2.1 Prototype Development

To start the development of FRED, there needed to be a test prototype in which the team could test preliminary ideas for the system, as well as learn how to implement the system effectively. Two versions of FRED were developed in order to do this, as well as to assess the perception and user experience of two different body designs. Both versions were developed with the same robot functions and verbal interaction. Version 1 (V1) was implemented with body movement (e.g., head movement) as the secondary feedback, and Version 2 (V2) with an LED face as its secondary feedback.

### 2.1.1 Mechanical Design

FRED's body is fully additive manufacturing using 3-D printing techniques, which is low-cost and resource efficient. Multiple FRED bodies could be produced without the creation of a specialized production line and could be made by anyone who has a 3D printer. In order to evaluate the perception of the two body designs, the same color and type of material was used for each body. The color chosen was a neutral gray color. Polylactic acid filament (PLA) was used for the FRED bodies, which is a low-cost, flexible filament that can be used for many different shapes. The appearance and size of FRED is shown in Fig. 2.1.

FRED V1 was the prototype that focused on the aspect of movement within its design. This robot body is comprised of three parts: the hemispherical base, the cylindrical neck, and the rectangular phone face. The hemispherical base is hollow in order to hold the electrical components of FRED. This base will be the foundation of the robot and ensure the structure does not topple easily while storing all the bulky components that make up the facilities of the robot. The cylindrical neck houses the servo for movement, and the rectangular head houses the phone. FRED V2 was the prototype that specialized in an LED face. This model consists of two parts: the large curved body and the conical-shaped head. The large body is half hollow, where the hollow section holds the electrical components, and the other half of the body contains a space for the phone to be securely placed. The phone is kept in place by a sliding door that is easy to remove. The conical-shaped head features an LED board that represents FRED's face.

### 2.1.2 Robot Operating System

FRED was created to be used on the Android platform through the user's phone, which would reduce cost by eliminating specialty hardware. Additionally, since the Android operating system has many features to handle input/output devices, such as the microphone, speakers, and camera, and a wide array of application programming

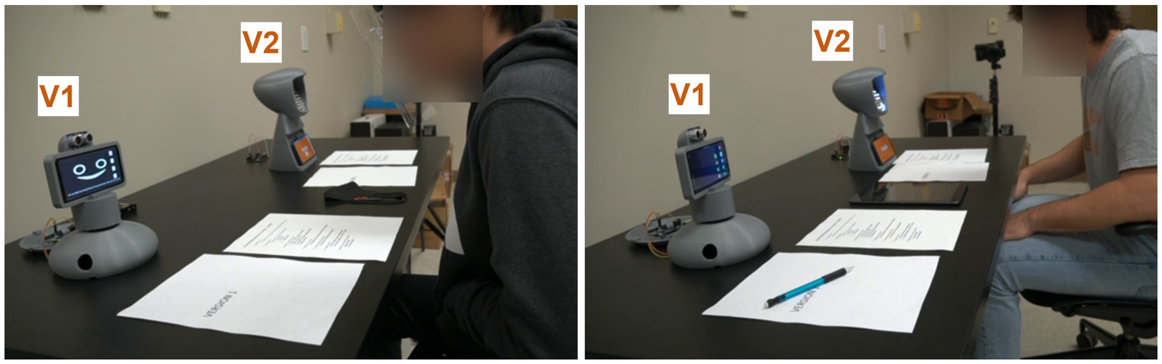


Figure 2.1: Participants interacting with FRED during a usability study. V1 and V2 can be seen in this figure.

interfaces (APIs), it is convenient to use an Android phone for development. Also, the automatic threading in Android makes it easy to achieve continuous speech recognition.

The phone is the central device in this system. An Arduino Uno, specifically the Wifi Rev2 model, was used to control the servo in V1 and the LED face in V2. Although we considered both Arduino and Raspberry Pi for this prototype, we chose the Rev2 in this project because of its wifi and bluetooth capabilities, as well as its smaller form factor than a Raspberry Pi. The smaller form factor is desirable, since this reduces the amount of printing required for the robot. The phone communicates with the Arduino through a Bluetooth Low Energy (BLE) connection, which allows the phone to communicate without a wire connection and create a more compact design. Users are able to control and interact with FRED through voice commands. Voice interaction eliminates problems with touch interface related to tremors or muscle weakness in users. The voice interactions are powered by an automated speech system that uses the speech recognition and speech-to-text libraries built into Android. The main software for the FRED system was developed on Android, mostly using the Kotlin language, as well as Java for the BLE integration, since the base code was already in this. Prototype 1's speech-to-text (STT) and text-to-speech (TTS) were run by Kotlin's built-in SpeechRecognizer and TextToSpeech classes respectively. These classes provided quick development to add these features, since these are complex features to implement in an application, even while using one of the multiple existing cloud APIs, like Google Compute Platform or Amazon Web Services.

### 2.1.3 Robot Hardware System

An Arduino is the ideal choice for controlling these peripheral devices, because the servo requires a pulse-width modulation (PWM) connection, and the LED ring needs a digital connection, which are both supported by the Arduino. Additionally, they are very affordable, with this model costing about $51.60 from Arduino's official website [27]. The Wifi Rev2 model was chosen specifically for its wireless capabilities to make the phone and Arduino modular, meaning the user may use the phone outside of the robot and still be able to connect to it in order to get feedback. Although a Raspberry Pi 4 has the same wireless capabilities for $35.00 [28], the smaller form factor of the Arduino was ideal, since both versions of the FRED robot have little space for internal hardware and wire organization. Additionally, if the wireless capabilities were no longer required, Arduino sells much cheaper models, like the Nano at $23.90 [29].

The phone communicates with the Arduino through a Bluetooth Low Energy (BLE) connection, which is the type of bluetooth connection supported by the chosen Arduino model. The low energy feature of this connection allows for a comparable connection speed to regular Bluetooth, but with considerably lower power consumption. As is pictured in Figure 2.2, the phone tells the Arduino which pre-programmed servo movement or LED face pattern to start by writing "characteristics" to the BLE connection, which are short strings of two characters that are defined by the programmer. For example, "01" on V1 represents the pre-defined servo movement that makes FRED shake its head to tell the user that they did not get the right answer. V1 has a small SG90 servo to create rotational head movement on the X-axis. This servo was chosen for it's low power requirements and affordability. V1 moves when users interact with it, for example, when a user says an incorrect answer in the game, it will shake its head as if to say "no, try again." V2 uses a 100-pixel NeoPixel disk to display a pre-programmed face pattern. Although Version 2 was planned to have different facial patterns for the experiment, this was not implemented before the experiment, so the robot showed a dynamic face with eyes and a mouth that did not respond to the phone.

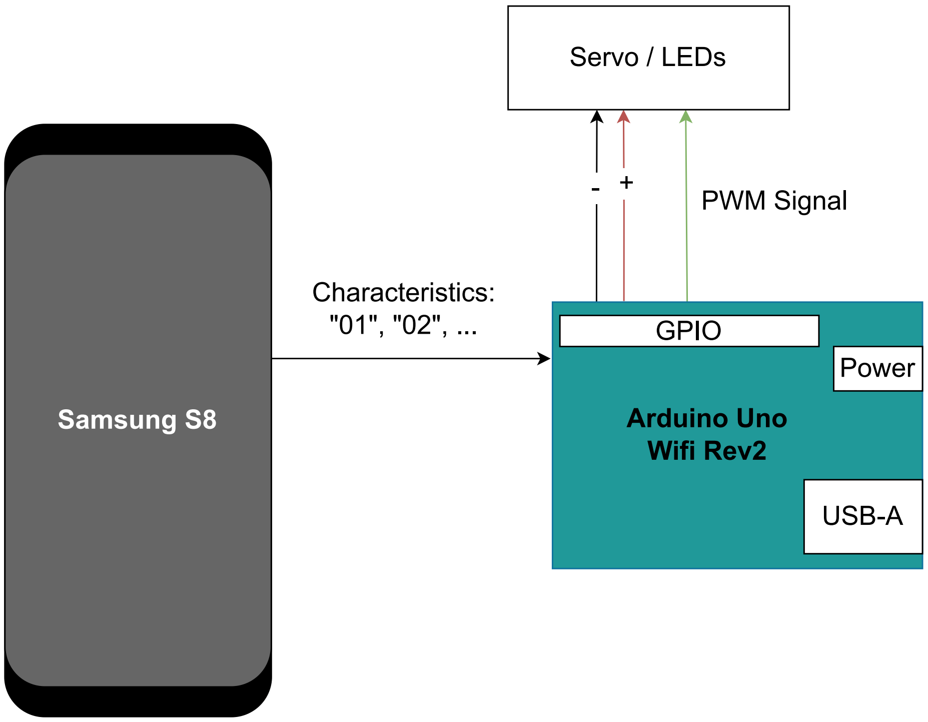


Figure 2.2: Communication between the Android phone, Arduino, and servo / LEDs. This communication is performed over a BLE connection. Based on the “characteristics” sent by the phone, the Arduino decides how to control the peripheral.

## 2.2 Arduino Development

Since the Arduino is a unique microcontroller, it is worth explaining some details on how to program on an Arduino controller. The Arduino Uno Rev2 is based on the ATmega4809 microcontroller, which is a very simple, 32-bit controller with a clock speed of 16 MHz, according to the Microchip Datasheet [30]. It is a 48 pin variant, and features 48 KB of flash memory as well as 1 KB of SRAM. Developing on these microcontrollers is ideal for controlling devices such as servos, LED strips, and other peripherals that require Pulse-width Modulation (PWM) connections. These types of connections are not readily available on a device like a smart phone, which is why the choice for this project was to use a separate controller. Programs for Arduino are developed in C. The Arduino does not have its own operating system, rather it runs any C program that is stored in its flash memory whenever it is powered on. This C program is already compiled before it is saved to the Arduino. These programs have a structure that is unique to Arduino. The first step of the program is to initialize any global variables declared outside of functions, as is normal in C. The setup() function will initialize any Arduino-specific properties, like the serial poll rate, the pin modes for whatever pins are being used in the projects, etc. Since Arduinos are event-based programming, just like mobile applications, there is an event loop, called loop(), that is called on each cycle of the Arduino. This is where code will be called repeatedly to check for new events and to handle them accordingly. Shown in Listing 2.1 is an example code snippet from the Arduino documentation of a simple program to check the value of a sensor and change an LED according to that sensor value. The setup() function is used to initialize the serial baud rate and the pin number to output, and the loop function is used to continually check the value of the sensor. This basic loop is similar to the way that the FRED Arduino checks for bluetooth characteristics, but more fleshed out.

```
//https://docs.arduino.cc/learn/starting-guide/getting-started-arduino#program-structure

int sensorPin = A1; //define pin A1 (analog pin)
int ledPin = 2; //define pin 2 (digital pin)
int sensorValue; //create variable for storing readings

//void setup is for configurations on start up
void setup() {
    Serial.begin(9600); //initialize serial communication
    pinMode(ledPin, OUTPUT); //define ledPin as an output
}

void loop() {
    sensorValue = analogRead(sensorPin); // do a sensor reading

    Serial.print("Sensor value is: "); //print a message to the serial monitor
    Serial.println(sensorValue); //print the value to the serial monitor

    //check if sensorValue is below 200
    if(sensorValue < 200) {
        digitalWrite(ledPin, HIGH); //if it is, turn on the LED on pin 2.
    }
    //if sensorValue is above 200, turn off the LED
    else{
        digitalWrite(ledPin, LOW);
    }
}
```

Listing 2.1: Example code snippet given by Arduino's website [31]

## 2.3 Features

The available activities for users was the same across both prototypes. These activities include the home screen, answering questions, creating contacts, making calls, and playing a shapes game. These activities were primitive shells of their fully featured counterparts due to time constraints, as well as testing. They were not meant to fully function as the intended activity would, rather, they remained sparse in order to test the usability and acceptance of the robot prototypes. Table 2.1 show all of the possible commands in this prototype. For example, the question-answering screen was only able to answer one hard-coded question ”When's the next UT football game?” As another example, the calling screen was a faked call with a ringtone in order to simulate what the call would look and sound like. The screens for these activities can be seen in Figure 2.3. These screens are primitive prototypes of the future functionality that FRED will have.

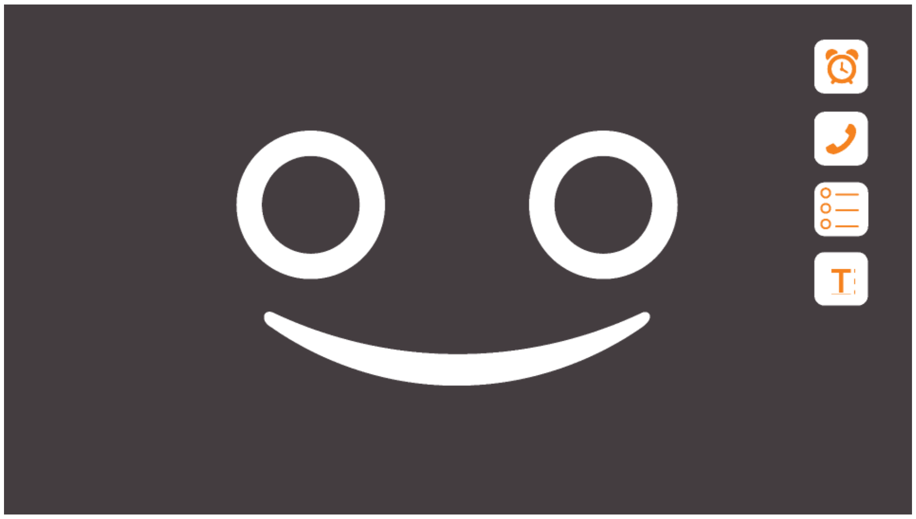
## 2.4 Testing: Usability Study

We conducted a pilot study to assess user acceptance, perception, and experience of the robot FRED through usability testing with young adults. Survey and observation approaches were used to learn users' perception of FRED's user interface (UI) design and functions. The study protocol was approved by the Institutional Review Board (IRB) of the University of Tennessee, Knoxville. IRB number is UTK IRB-21-06480-XM. We recruited potential participants via emails. The participants were students recruited from the University of Tennessee, Knoxville. Eligible participants included adults who met these criteria, 1) aged above 18, 2) able to understand and speak English fluently, and 3) no severe hearing or visual impairments that would interfere with interacting with the robot FRED.

Table 2.1: Voice commands to start each task.

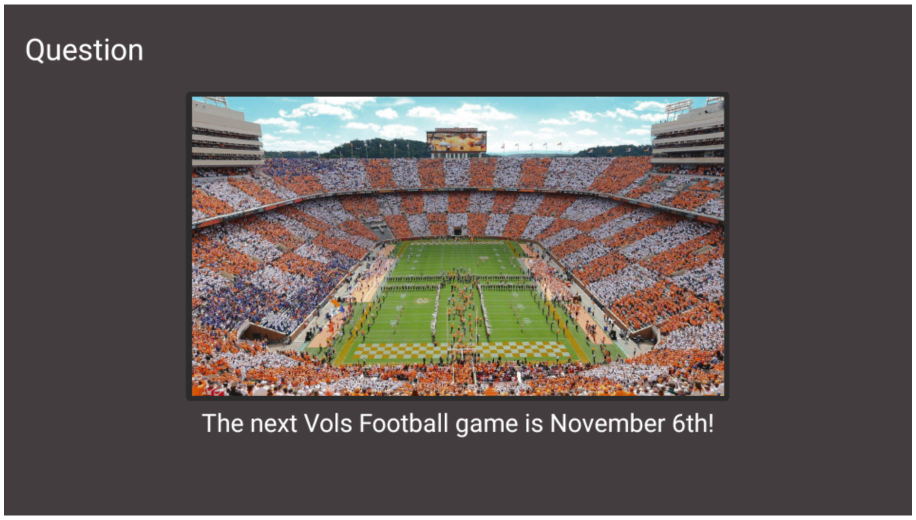
| Task | Command |
|---|---|
| *Start FRED* | “Hey FRED” |
| *Create a contact* | “FRED, I’d like to create a contact.” |
| *Make a call* | “Please call Tom.” |
| *Set a reminder* | “FRED, remind me to drink water every 15 minutes” |
| *Inquire football time* | “FRED, when’s the next UT football game?” |
| *Play a game* | “FRED, I’d like to play a shapes game.” |
| *Move head* | “FRED, look left.” or “FRED, look right.” |

(a) Loading screen



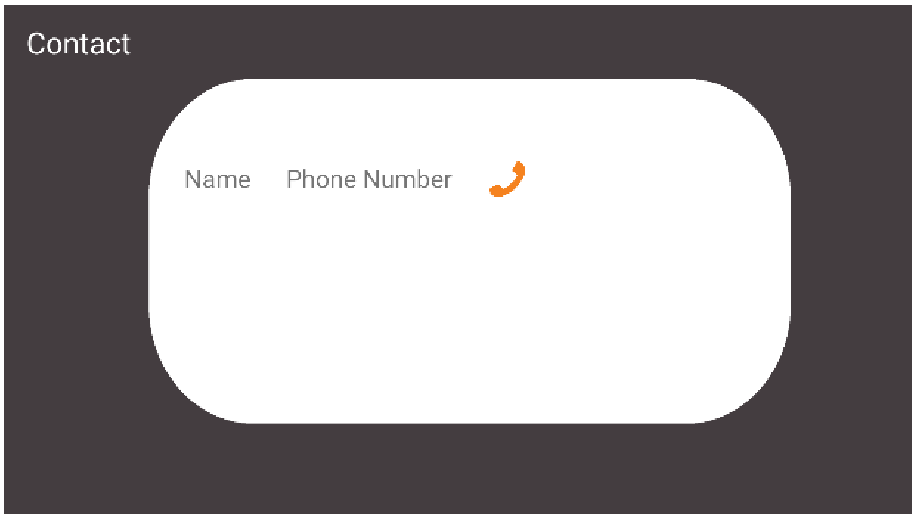
(b) Home screen

Figure 2.3: Examples of graphical user interface on the phone of FRED.



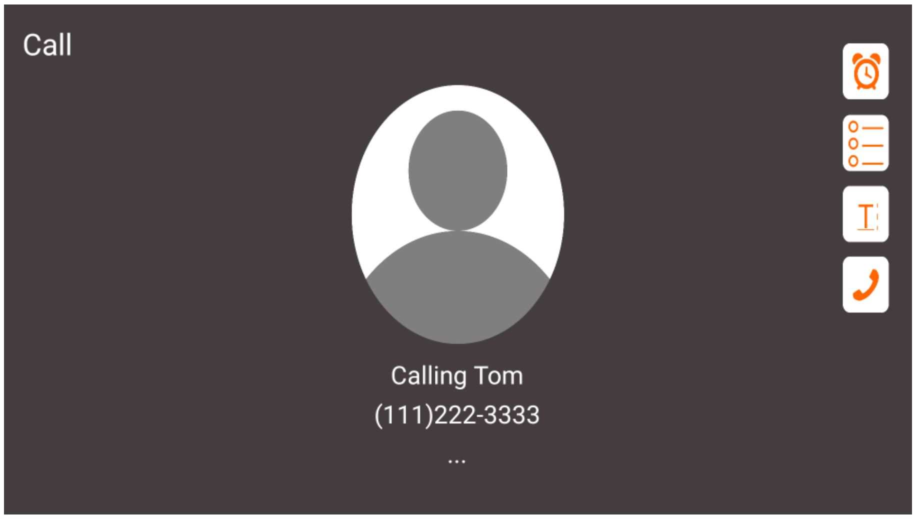
The next Vols Football game is November 6th!

(a) Question screen

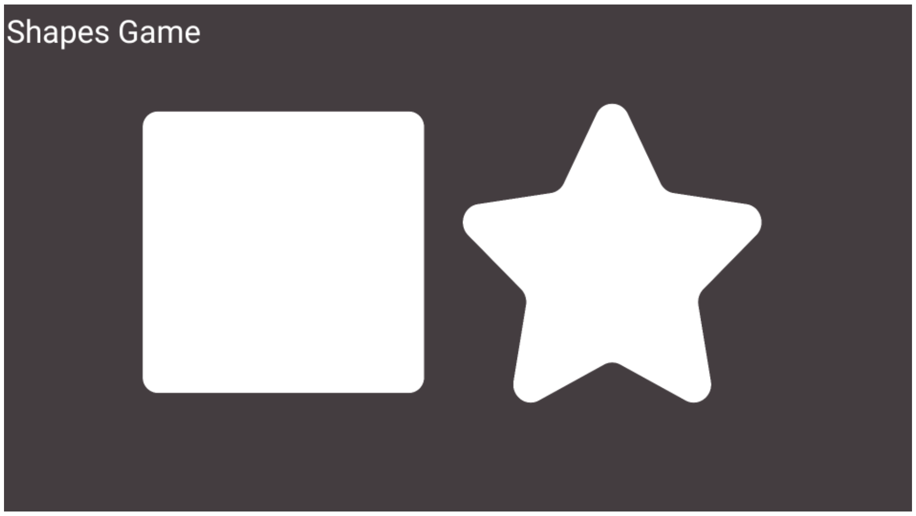


(b) Create a contact screen

Figure 2.3: continued



(c) Call screen



(d) Shapes game screen

Figure 2.3: continued

### 2.4.1 Data Collection and Analysis

The post-test survey consisted of 19 items to evaluate participants' perception and user experience of FRED, for example, users' positive response to its UI design (i.e., appearance, voice qualities, and height) and ease of use. Six items, including five questions on a five-point scale and one open-ended question, were specifically targeted at V1 and V2 separately. The other seven items were applied to both prototypes.

Descriptive statistics including frequency, mean (M), standard deviation (SD), range, and proportions were used to analyze participants' responses to the survey. Considering the small sample size, we manually went through all the comments from open-ended questions to gain an understanding of participants' experience with the two versions of FRED. As for the recorded videos, we used Simple Video Coder [32], a free, open-source software for efficiently coding social video data, to code participants' interaction behaviors during the experiment, including the participants' emotions (such as laughter/giggling/joy, frowning, and confusion) and robot-related failures during the tasks.

### 2.4.2 Results

The demographic survey results can be seen in Appendix A, Table 1. Table 2.2 shows participants' responses to the post-test survey regarding each FRED version. Q1 shows that participants, on average, *probably* wanted to have a robot in their homes to assist them with daily activities. Q4 and Q5 are similar, showing that participants *probably* wanted FRED to be their friend and would follow advice provided by FRED. On average, users felt like they would be more likely to be able to use V2 without any help with a mean difference of 0.93. Users also felt more afraid that V1 was going to break during use. Q6-8 are specific questions about what users prefer. Users preferred the current appearance of V2. Although robot's voice was the same for both, users slightly preferred V2's voice. Users did not report any preference of one height over the other, with a minor difference of 0.07 in the mean rating to FRED's height (Q8 in Table 2.2).

In examining the coding results for experiment videos, Version 1 had many more robot-related failures than Version 2, a total 70.7% of the failures. V1 had 87.5% of all instances of user laughter during testing. It was found that all codes of confusion, frustration, and shock came from testing with V1, either from the head movements or speech recognition failures.

## 2.5 Next Steps

The built-in TTS provided acceptable performance, although the voice was robotic. The STT, however, did not perform well. It would often misunderstand the user's utterance, or stop recognizing altogether. It was clear that another solution would be necessary. One advantage of developing for Android was the ability to develop quickly for multiple screens, as well as the ease of deploying the application to any Android device. Unfortunately, the application would behave differently for different phones, which would create problems in the future for maintaining the software for users. Additionally, the phones being used were already cost-prohibitive at about $200 each, which did not leave much room for other future additions or more complex 3D-printing. It also did not allow any choice as to what specific peripherals were being used, such as the camera.

In both versions, the Android phone communicated with the Arduino through a bluetooth connection, which avoided the addition of a wired connection. The original intention of this was to reduce the complexity of the 3D-printing and assembly process, however, this increased the software complexity and reduced the reliability of the system. During testing, the phone would often lose connection with the Arduino, leading to frustration from users and many pauses in the experiment for troubleshooting. This also introduced latency, which would inhibit more complex, real-time communication between the devices on future features.

Table 2.2: Post-test survey results

| Question | Statistic ($M \pm SD$) |
|---|---|
| Q1: Do you think you would ever like to use a robot to assist your daily activities or to keep you company? | 4.20 $\pm$1.08 |
| Q2: Would you like FRED to be your friend? | 4.07 $\pm$1.22 |
| Q3: Would you be likely to follow the advice provided by FRED? | 3.87 $\pm$0.99 |
| Q4: Did you feel like you could use Version X FRED without any help? | V1: 3.40 $\pm$1.24<br>V2: 4.33 $\pm$1.05 |
| Q5: When using the Version X FRED, were you afraid you would break something? | V1: 1.53 $\pm$0.52<br>V2: 1.00 $\pm$0.00 |
| Q6: How likely do you think people would be to use a Version X with its current appearance? | V1: 3.07 $\pm$0.70<br>V2: 3.60 $\pm$0.99 |
| Q7: How likely do you think people would be to use a robot which has the current sounding voice as Version X? | V1: 3.07 $\pm$1.16<br>V2: 3.40 $\pm$0.99 |
| Q8: How likely do you think people would be to use a robot which has the current height as Version X? | V1: 3.80 $\pm$0.78<br>V2: 3.73 $\pm$0.88 |

*Note:* Q1-4: 1 = No, 2 = Probably not, 3 = Maybe, 4 = Probably, 5 = Yes.
Q5: 1 = No, 2 = A little bit, 3 = Yes.
Q6-8: 1 = Very unlikely, 2 = Unlikely, 3 = Neutral, 4 = Likely, 5 = Very likely.

# Chapter 3

# System Design

The second prototype of the FRED system needed to be more robust than the first. It was clear that the initial platform chosen, the Android phone and Arduino, was not sufficient for this project. Therefore, this second prototype would involve a total overhaul of the system's platform, more mature features that could be used by an end-user, and software design that could be long-lasting in development.

## 3.1 Hardware Platform

### 3.1.1 Hardware

Based on the results from the first prototype, it was clear that a change in platform was necessary in order to speed up development time and create a more robust system. Therefore, a Raspberry Pi programmed with Python was chosen for this role. Shown in Figure 3.1 is the current prototype of the hardware configuration, which was initially designed and implemented by the senior design team in 2021-2022. The team members were Cayse Rogers, Yunfei Liu, Houston Lambert, Ty Banks, and Quintrell Payne. This design has been slightly modified, since the original used a Raspberry Pi3, but the current design uses a more powerful Pi4. The Pi is connected to the microphone and touch screen through USB connections. The touch screen requires a

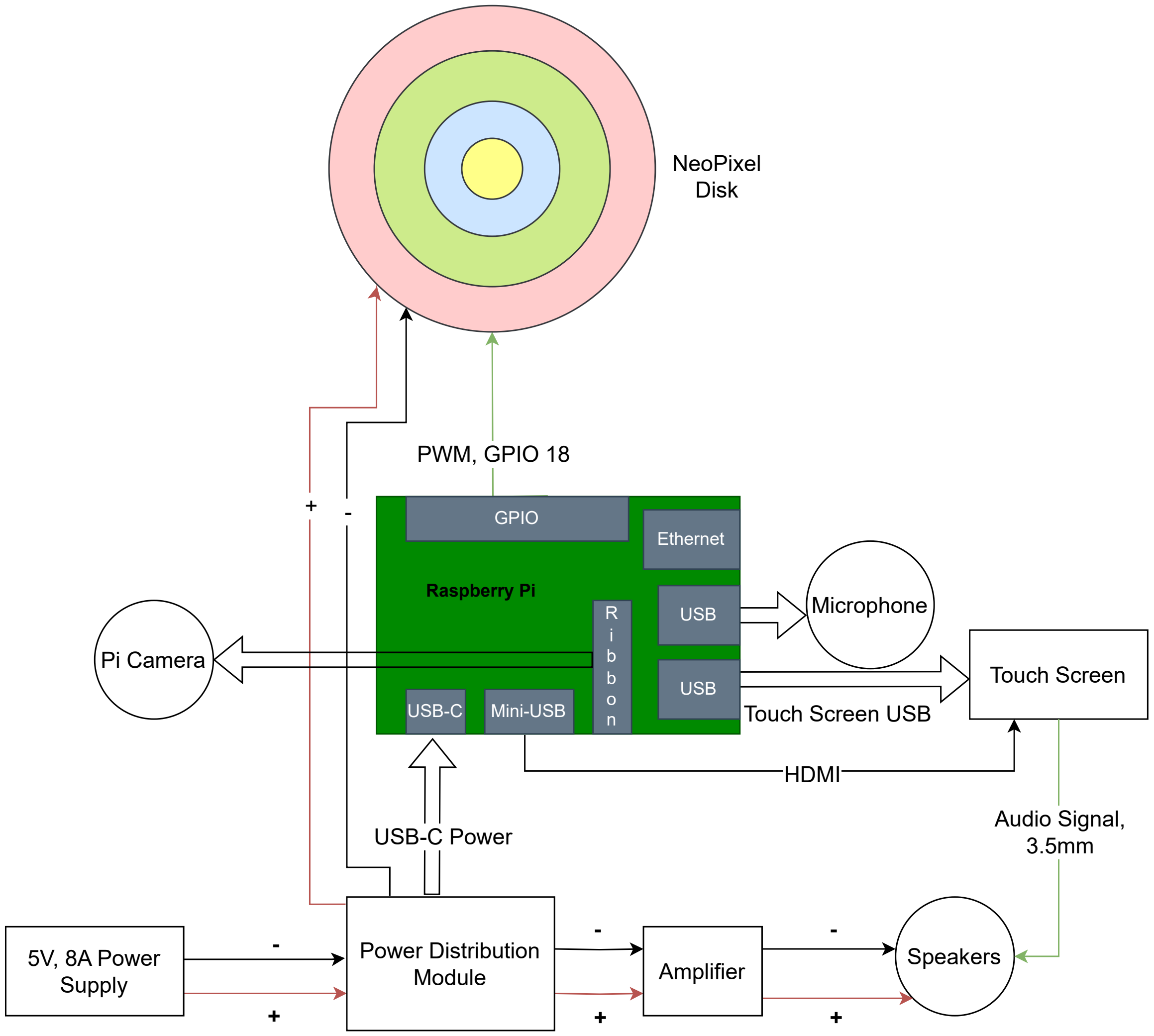


Figure 3.1: The current hardware configuration of the Raspberry Pi and all its peripherals.

USB connection for both power and communication for the touch capabilities. It is also connected to the Pi through HDMI to display the screen contents. Additionally, the Pi is connected to a camera, although the camera is not currently used by the FRED application.

This configuration features a central power distribution module that provides power to the Pi, the LEDs, and the amplifier. This ensures that all of these units receive the amount of power that they require. The power supply used is a 5V 8A AC-DC power supply. The 8A is more than enough for this project, but it also allows future expansion of peripherals, should such a change be warranted by user needs. 5V is required by the Raspberry Pi in order to function properly, and the rest of the modules work with 5V supply. The amplifier is necessary for the speakers, since the Pi does not have a powerful enough signal to power these speakers alone, rather it has enough to supply signal to a pair of low-resistance headphones. The audio signal is supplied to the speakers through the touch screen, which has its audio routed through HDMI. Although this is inelegant, it is necessary to work with the LED disk, because the NeoPixel library for Raspberry Pi and Python requires that audio through the 3.5mm port be disabled. It is not clear through their documentation why this is necessary. Notice that there are no extra controllers that perform certain tasks and are communicated to by the Pi. The beauty of this configuration is that the Pi handles all controller tasks, such as changing LEDs, delivering audio and video, and managing user input through a microphone. This reduces cost and complexity of the system.

## 3.2 Android

One simple solution to the Android phone problem was to use a Raspberry Pi 4 with an install of Android 11 OS via Emteria [33]. This solution would allow the reuse of the current code base that was already programmed for Android, while using a more affordable and flexible computing platform. The first problem that was clear upon implementing this was the amount of overhead that Android OS required to run. The app loaded slowly, transitions between screens were significantly delayed, and any problems with STT were amplified. Additionally, no proprietary Google features were available, which meant that using built-in TTS voices were not possible by default. Many of the requirements to set up the Pi with Android in order to simply run the application meant that this would be very time-consuming to reproduce for multiple robots.

### 3.2.1 Raspbian

The previous platform of the Android phone and Arduino left much to be desired. The bluetooth and speech capabilities were unstable, and integrating cloud technologies, like GCP and OpenAI, were difficult and unreliable. Additionally, the code is all based in Kotlin and Android-specific functions and classes. If Android became obsolete for this robot application in the future when the code is already well-developed, the platform switch would be a long, arduous process. For this reason, we wanted to switch to a platform that has many libraries available for cloud computing, and is able to be run on many different systems. For this reason, we chose to switch to developing on Python. It has a plethora of libraries available, especially for cloud-computing services and machine learning. It is also quick to develop on, which would allow more rapid prototyping. Lastly, it is simple to create all of the specific library version requirements for any python application, so it is easily portable to any system, including laptops, desktops, and micro-controllers. This also allows more extensive testing, since these devices are all readily available, and more uniform in behavior.

## 3.3 Software Features and Purpose

In order to create a more capable product, the goal of Prototype 2 was to develop meaningful features that are fully developed, as opposed to the primitive activities seen in the first prototype. The FRED robot features capabilities for providing people with ADRD company, information, and reminders to improve their quality of life. FRED is able to maintain a conversation with its users in order to create a sense that it is a companion that they can talk to. FRED is also able to provide reminders for events, like doctor's appointments and taking medication. It also serves as a way to store important information, like contacts for the user's family members and caretakers.

FRED currently features six different screens and four activities for the user to choose from. The first screen the user is greeted with is the login screen. It is a simple login like many other applications, in which new users will create a new account with their email and password, and returning users will use their existing login. Although this is the current solution, it is a clunky design, since the user will have to use the on-screen keyboard to perform this. A possible future solution is a facial recognition system that allows users to login with ease. After logging in, the user sees the home screen, which serves as the origin for all activities, and is the section of the FRED app in which the robot will freely have a conversation with the user for as long as they want. The home screen features large buttons that all redirect to different activities. It also shows the current date and time, and a placeholder for the current weather which is currently hard-coded. FRED has four different activities to choose from in the home screen: creating and viewing contacts, a check-in on the user's mood that day, a story game, and a reminder screen for creating and viewing events. The screen also shows the buttons for activities that are soon to be developed.

The contacts screen is where users create and view their FRED contacts. The user is able to input the first and last name, address, and phone number of the desired contact. Each contact is given a default picture. These contacts are all solely made via FRED and stored on Realtime Database. They are not connected to their mobile devices. The hope is to create a unifying system for contacts through an API such as Google Contacts in the future. Additionally, a future feature will be to be able to edit contact photos to the desired picture. Lastly, the ability to make phone calls to these contacts will be added.

The check-in screen is a simple activity where FRED prompts the user by saying "I'd love to know how you're doing today. Touch the face that matches your mood today, or say the mood that's under the face." There are five faces on the screen, which mimic a Likert scale in order from worst to best: Horrible, Upset, Okay, Good, and Happy. This activity has multiple purposes. The first is to make the user feel like FRED cares about how they feel and to gain trust from the user. The second goal is to track the user's feelings throughout the day, week, month, and so on for their caretaker or family. It is important to track how the person with ADRD is feeling, and this activity will enable statistics to be drawn about the user's feelings throughout any time period. A caretaker or family member will be able to see any possible trends in their feelings and create a plan to mitigate the problems they are experiencing.

The story game screen is a game-like activity for the person with ADRD to use their imagination. The screen shows a large image and a text box that shows what the user has said. FRED prompts the user by saying "I'd like you to tell me a story about the image shown on my screen. Say the words "I'm done" when you're finished." The user can respond however they want, either by creating a story that involves the image or by simply describing the contents of the image. FRED then says "Thank you for telling me a story about this image! I'd love to talk about it more with you...", and will begin asking the user questions about the image. After the user responds to each question, FRED will provide some feedback on what the user said. These questions and feedback are all generated by OpenAI. Some problems naturally arise from using this OpenAI approach, which will be discussed in Section 4.6.

The reminder screen enables users to create and view reminders for important events, like doctor's appointments, family events, and taking medication. This is the most-requested feature among users. In order to create a usable interface for people with ADRD, the reminders screen will have multiple viewing modes, which include a daily view, weekly view, and calendar view. The daily view shows users what events the current day holds, while the weekly and calendar views give larger insights into what the events are for the future. The default view is the daily view so that users are not overwhelmed with the amount of information available in other modes. Users are able to create events for a given day by giving a title, a start hour, and an end hour. These reminders are pushed to the user's Google Calendar, where they can view them from any device in addition to FRED.

### 3.3.1 Interaction with FRED

Users are able to interact with FRED through touching the tablet screen in the center of its body, as well as through speech. FRED is capable of performing STT and TTS, so users are able to freely converse with the robot. Additionally, users are able to have conversations with FRED through the power of GPT-3 using the OpenAI API. Users can simply have a conversation with FRED, in which it is providing dynamic responses, or they can play a game with it, such as the story game, in which users tell FRED a story about the image shown on its screen, and FRED will ask the user questions about it. An example of this interaction can be seen in Figure 3.2, where the user is creating a contact with FRED. These interactions create a richer experience that make the robot more than a tablet.

### 3.3.2 Storing User Data

The safety of user data that is stored in FREDs software is important, as the users have waning memory. If data is lost, such as their contacts, it will be confusing and disruptive to them. In talking with users at the focus group performed, they are concerned with anything changing in the application, like the location of buttons and data. If their contact data is lost, this could be disorienting to them. Therefore, it is important to store their data in a secure cloud solution, since their data will not be

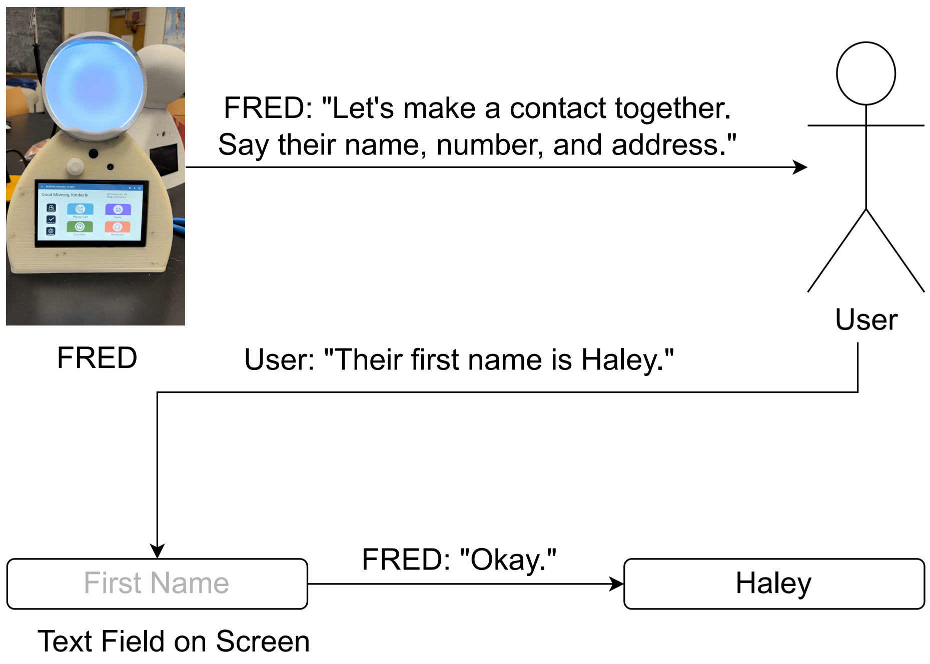

FRED: "Let's make a contact together.
Say their name, number, and address."
User
FRED
User: "Their first name is Haley."
First Name
FRED: "Okay."
Haley
Text Field on Screen

Figure 3.2: Example interaction between the user and FRED on the contact creation screen.

subject to any local problems. Many problems can occur, such as power loss, while reading or writing to files, so this would not be the most ideal solution, especially when users may unplug FRED or turn it off unexpectedly.

## 3.4 Software Design Patterns

Design patterns are a useful tool for creating more flexible code that can be reused and more easily adapted. It is important for software to be flexible, because technology evolves so quickly. What was previously the best cloud storage method can quickly become obsolete. Design patterns allow software designers to create separation between platform-specific code that will handle a certain task, like displaying the user interface, and the control logic of the program, which makes the code much easier to adapt when a dependency needs to change.

### 3.4.1 System: Model View Controller

The Model View Controller (MVC) design pattern is useful to this project because of the separation between classes that it provides, which is important for future platform changes. In this design pattern, the Model represents the data and business logic of the current screen, including data validation. The View class handles all of the visual components that the user is interacting with on the screen, including buttons, text fields, and scrollable areas. Lastly, the Controller class acts as the mediator between the other two classes. In the traditional MVC definition, as described by Krasner et al., the model and the view are much more interactive with each other, as can be seen in Figure 3.3 [19]. This creates an automated interaction upon changes in the data, but it also creates more work to add new features. This can be troublesome for development. The system used in this work will use the controller as a central worker for modifying data. In order to create a better development environment, a less-coupled version was used, as seen in Figure 3.4, in which a user interacts with

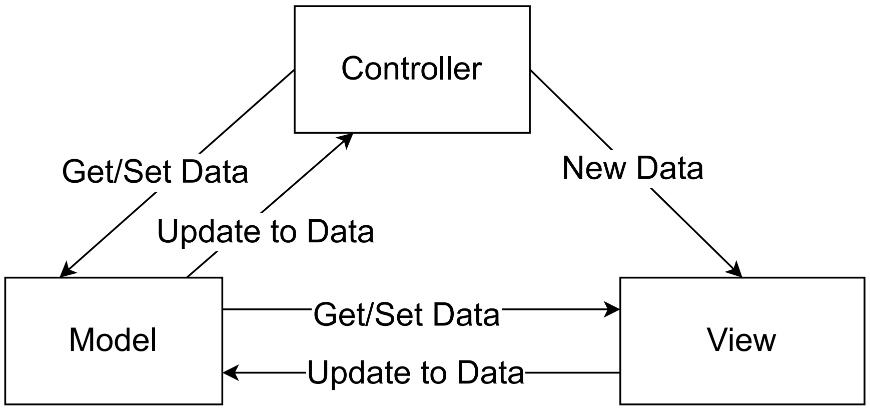


Figure 3.3: MVC definition as described by Krasner et al. [19]

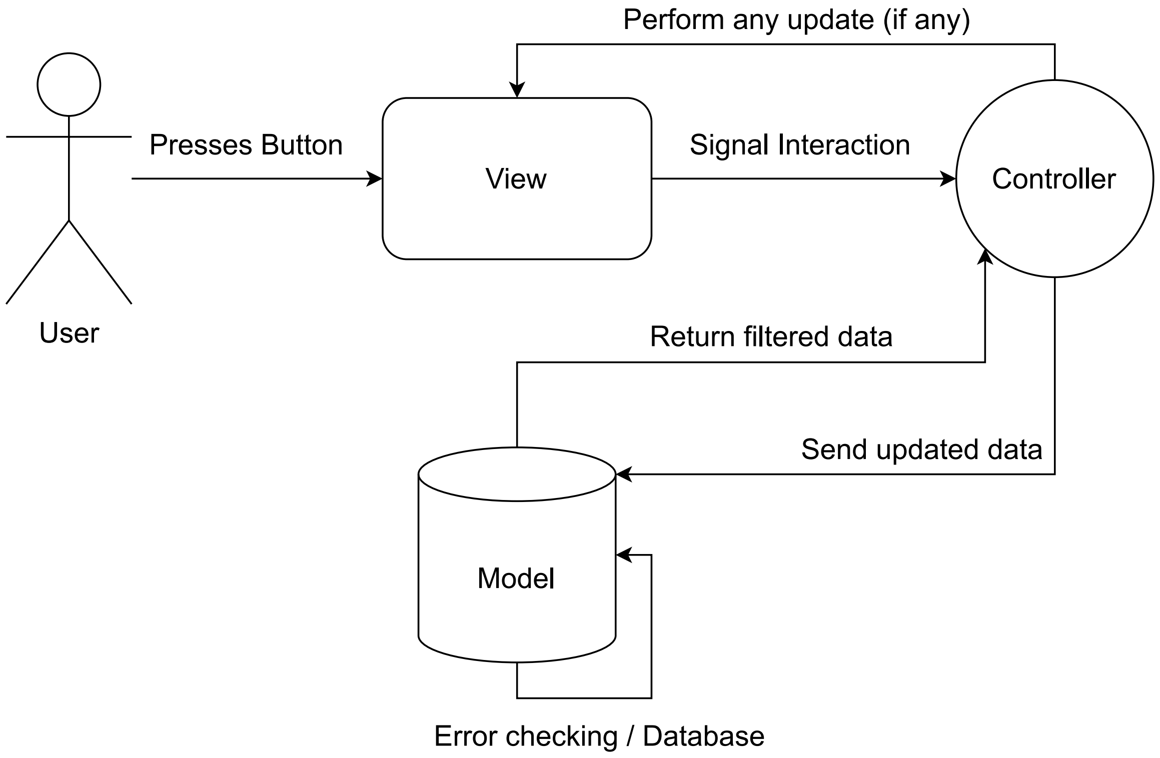


Figure 3.4: Interaction between user and MVC system.

the view and the system updates throughout. In this version, updates to either the view or the model must go through the controller. This ensures that the controller is the central hub for information, creating a more central place to modify code, while also keeping abstraction between the model and view. Additionally, this means that the model and view do not need to know anything about each other. They act as passive classes that are used by the controller in order to do some task, while also performing important tasks in the background, like data verification for the model and event-handling for the view.

The model class handles all of the details of the data to be maintained in the current screen. This includes creating, reading, updating, and deleting (CRUD) data between a database. In this project, the CRUD operations are handled by the Data Mapper classes, talked about in Section 3.4.3. However, the model handles the interaction between the controller and this mapper class, as well as any other data that needs to be stored, such as a contact's name or phone number. This interaction between the model, controller, and database can be seen in Figure 3.5.

Each view class handles only the specific tasks to displaying information on the interface and handling user input events, like button presses and text input. In order to keep this agnostic from the controller's function and variable names, the view is given controller-defined functions for handling events in its constructor through dependency injection. The view then saves references to these functions and calls them when the event occurs. This allows function refactoring to occur only in the controller or view respectively, without the need to change both. In the case that a platform change for the GUI were to become necessary, the changes would only occur in the view class, which will increase the flexibility of the program. The abstract view class, shown in Figure 3.6 inherits from the Subject class, which gives it the ability to update its observer, in this case the controller.

The controller is the central brain of the operation which handles what should happen to the data in the model on each event that occurs from the view. It also acts as a mediator between the view and the model in the sense of gathering data, since

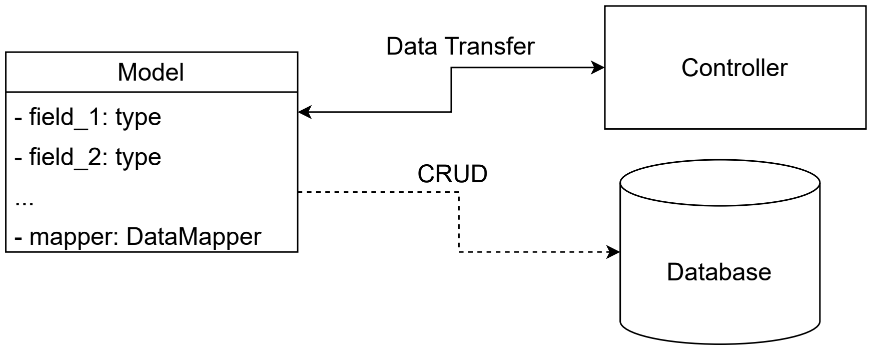


Figure 3.5: UML diagram showing the interaction between the model, controller, and database.

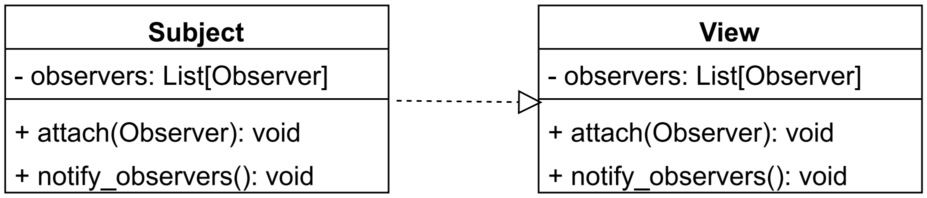

Subject
- observers: List[Observer]
+ attach(Observer): void
+ notify_observers(): void
View
- observers: List[Observer]
+ attach(Observer): void
+ notify_observers(): void

Figure 3.6: Abstract UML diagram of the View class inheriting from the Subject class.

the view must ask the controller to get or modify piece of data, which the controller then requests from the model. This means that changes to the model will always incur a change in the controller alone, whereas, if the view understood what is inside the model, the change would need to occur between the model and view class, which creates more dependency between these two classes.

The controller employs multiple inheritance in order to be a child of both the GenericScreenController and Observer classes, as is shown in Figure 3.7. This is necessary in order to communicate with the view efficiently, since an Observer-Subject pattern is employed here to create easy updates from the view. On an update to the screen, the view will notify its observers, the controller in this case, that there is new data to be extracted and handled. The controller updates through its update() function, which is defined individually for each screen, since there is different data to handle on each one. This allows a more flexible update that is less prone to error, since the controller will always grab multiple pieces of information from the screen in its update function, as user interactions can often cause multiple changes. For example, say the user presses the "Create Contact" button on a contact-creation screen. In this case, the controller needs to collect all of the information the user entered for the contact, which includes their first name, last name, phone number, and address.

### 3.4.2 Updates: Observer-Subject

The Observer-Subject pattern is an easy method for communicating new information between two different classes. Figure 3.8 shows the abstract implementation of this design pattern. In this pattern, the Subject is a class that contains information, a list of observers, and a method for updating its observers. The abstract class defines a method called attach(), which will add an observer to the subject's list of observers. The notify_observers() method will iterate through the subject's list of observers and call each one's update() function. The observer contains a variable to store its subject and an update() function. The update() function will be called by

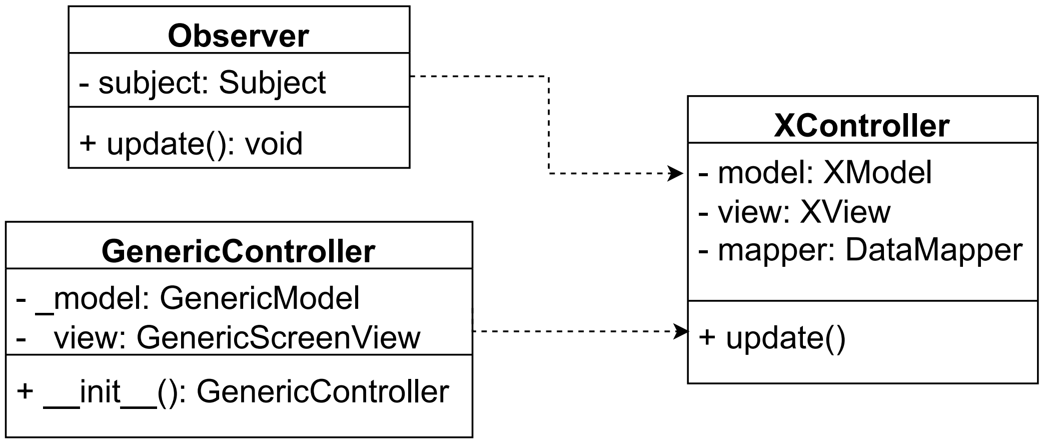


Figure 3.7: UML figure showing the abstract implementation of the controller.

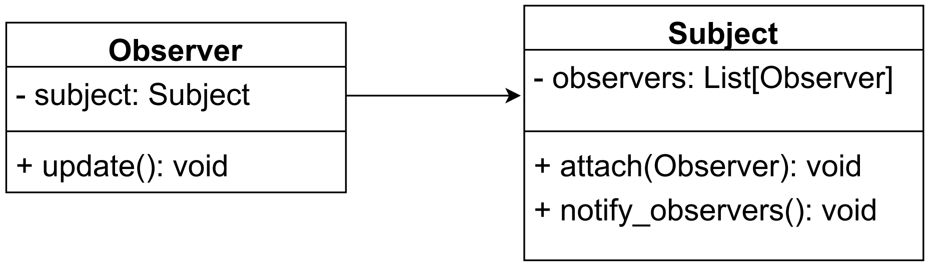


Figure 3.8: Abstraction of the Observer-Subject design pattern.

the subject, and it will contain specific implementation on what information to obtain from the subject. For example, if the subject stores data on the current contact being created in the program, the observer will grab those specific fields, like the first name and phone number, in order to do some operations on those for itself. The pattern itself is simple enough that it can easily be changed to have different classes be the observer and subject. For example, if it is desirable for the model to be the subject of the view in order to have automatic updates to its fields, this is possible under the framework. In Chapter 4, it will be shown why this was not chosen for this particular implementation, but it is certainly available if it suits the programmer's needs.

### 3.4.3 Database: Data Mapper

The Data Mapper pattern is used to abstract the create, read, update, and delete (CRUD) functions of a particular data class, which can be seen in the abstract UML diagram in Figure 3.9. CRUD functions are the foundational operations for handling database entries. The data mapper creates a layer of abstraction between the database implementation and the functions that need to save information to the database. Whenever a new object needs to be created and pushed into a container or a database, the actual implementation of this object can be simple and intuitive, while the data mapper can be as complex as it needs to be in order to save it in the database. In the case that the database platform needs to change, adding this layer of abstraction will allow a simple change to the classes that are saving information. This pattern is not only able to be used in simple database operations, it can also be used for abstracting long, complex API calls in which data needs to be stored. An example of this is a calendar API, which will be shown in detail in Chapter 4. Although this pattern is simple, it is also versatile in the operations that it can be applied to.

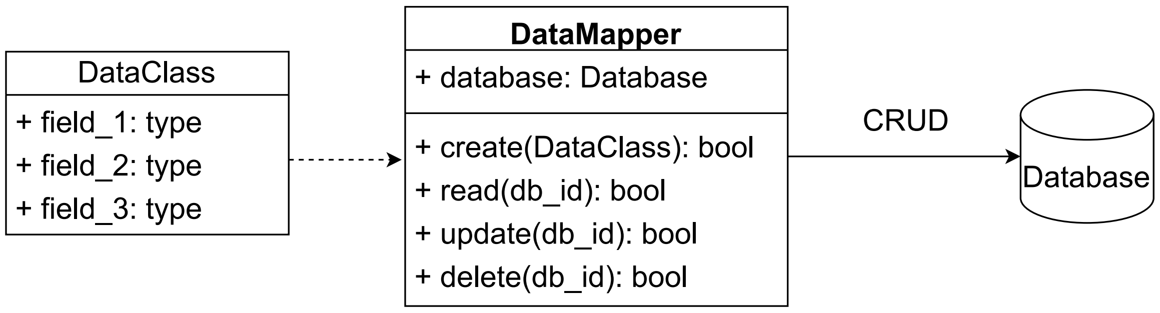


Figure 3.9: UML diagram of abstract data mapper class.

# Chapter 4

# System Implementation

## 4.1 Developing on Raspberry Pi

The Raspberry Pi is a much more capable controller than the Arduino. According to the Raspberry Pi documentation specification sheet, the Raspberry Pi 4 Model B used in this project is based on a Broadcom BCM2711 CPU, which is a quad core, 64-bit processor [34]. The Pi features its own dedicated Linux operating system (OS) based on Debian called Raspbian, which also features its own desktop environment. Additionally, the Pi is an ARM v8 system, which can limit compatible Python modules, like cefpython. The Pi features 1 - 8GB of SDRAM and a 40 pin GPIO header, which allows for many different peripherals to be used on the system. Since the Pi has its own OS, GUI applications can be run natively, either in its desktop mode or in “kiosk mode” through the commandline. The kiosk mode is preferable, since there will be less boot time involved. Additionally, the Pi is able to have its own Python interpreter, so any Python program with Pi-compatible modules can be run from its terminal. This creates a much friendlier research platform, because it uses a higher level programming language that is easier for researchers that do not have a computer science background, which is common in ADRD-related social robotics. It is also much more accessible, since Pis are traditionally readily-available and affordable. Although a trend has started where the opposite is true, other boards are commonly available that achieve similar results to the Raspberry Pi.

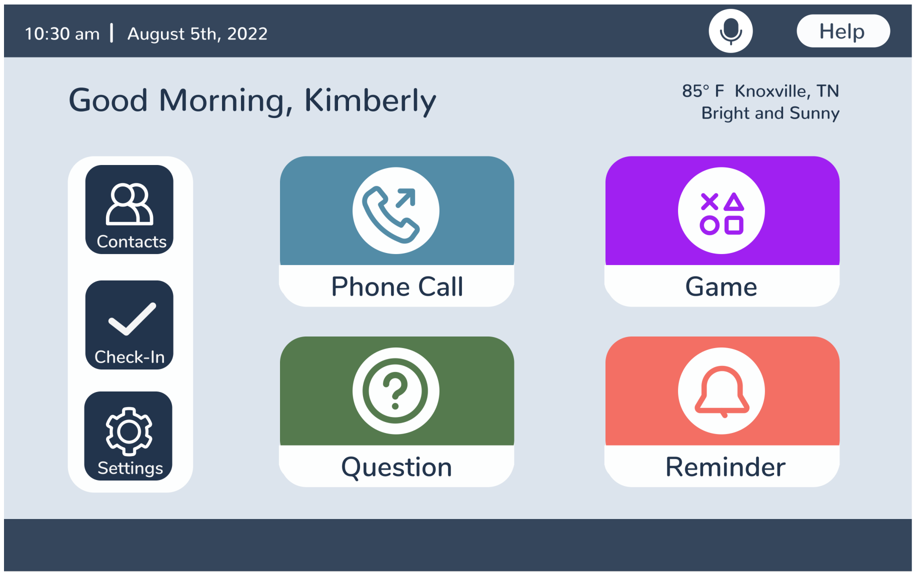
## 4.2 Kivy Suite

The Kivy and KivyMD libraries were chosen to develop the GUI. These libraries offer modern-looking app components and a simple architecture that matches Android well. These libraries are intended for mobile application creation, so they are able to be compiled for Android and iOS. Additionally, they have simple syntax that mimics the Android XML layout files in a more concise way. Unfortunately, one disadvantage to developing on this library is a lack of live editing support. Although this feature is available for some systems, it does not consistently work on some, like Windows machines. This means development time can be slower if intensive GUI development needs to be done. Another drawback is that some features are limited in customization. For example, the colors for elements like the MDTextField are not all able to be customized. This leads to problems in uniformity among UI elements. Examples of the GUI can be seen in Figure 4.1. These screens were designed to be easy to understand and manipulate by older adults with mild cognitive decline.

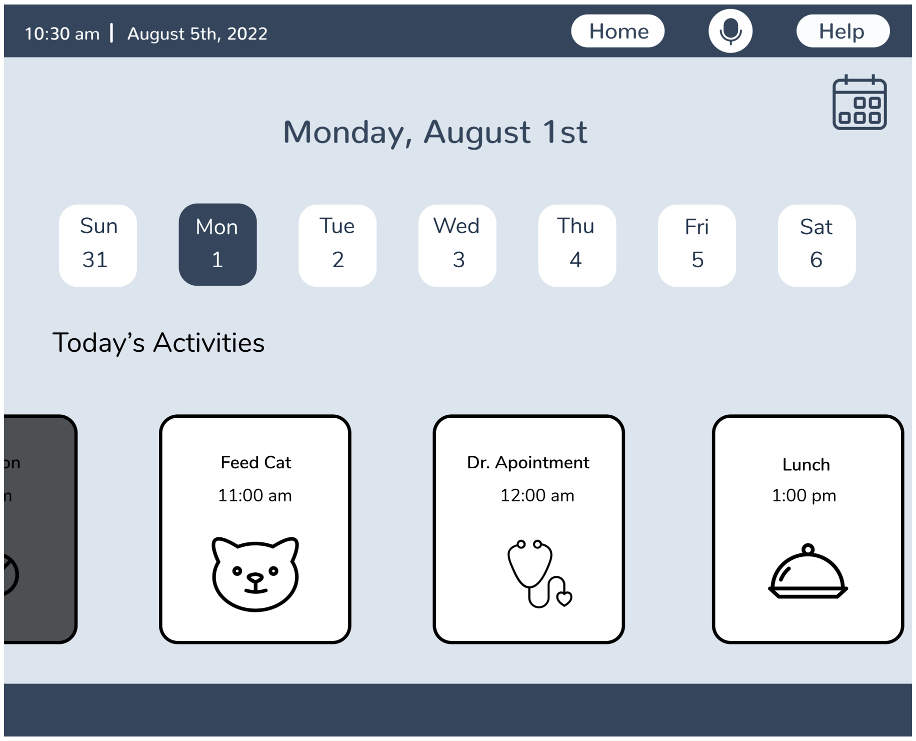
## 4.3 Design Patterns

### 4.3.1 Model View Controller

In traditional definitions of the Model View Controller (MVC) pattern, the user performs some action to the screen, like clicking a button, and an event is sent to the controller to be handled. The controller then modifies the data in the model and the model notifies the view that there has been an update. For this project, this pattern has been adapted to be create less-coupled classes. Instead of the model updating the view, the controller directly modifies the view based on the resulting data in the

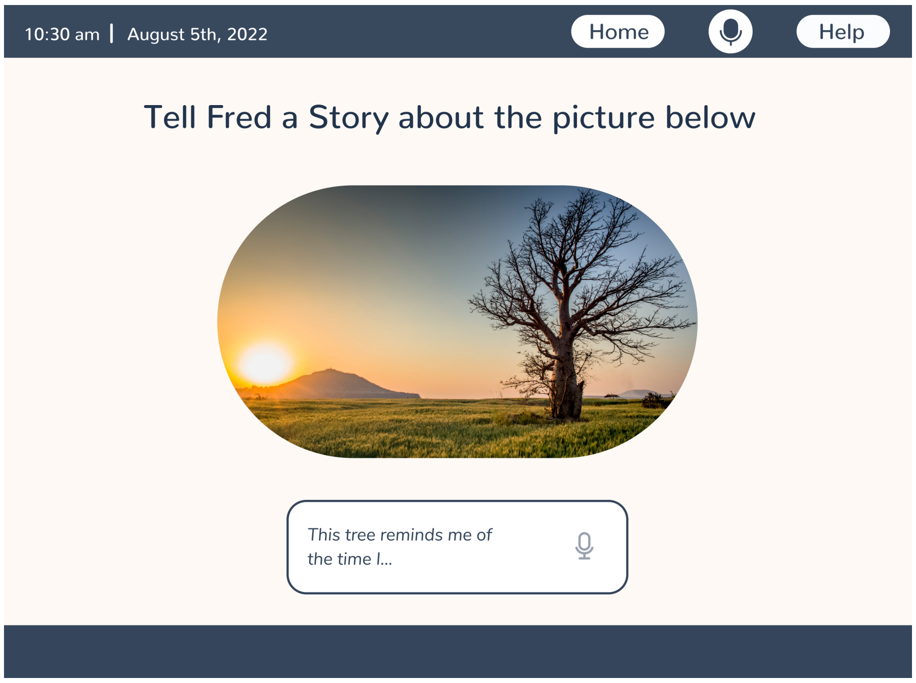


(a) Home screen

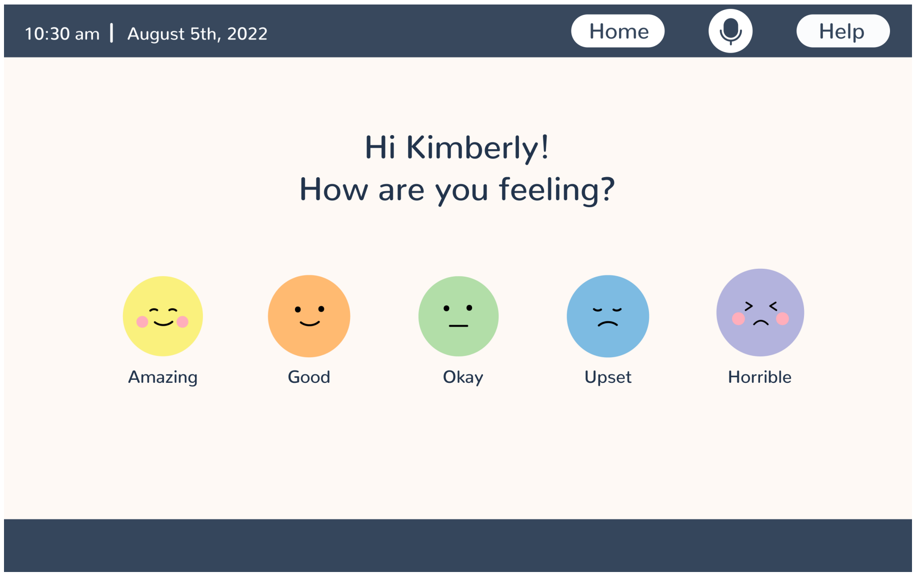


(b) Reminder screen

Figure 4.1: Example design images of activity screens


10:30 am | August 5th, 2022
Home
Help
Tell Fred a Story about the picture below
This tree reminds me of the time I...

(a) Story game screen



(b) Check-in screen

Figure 4.1: continued

model after the action. In this way, the model and view are both agnostic from each other, allowing straightforward decoupling. Additionally, this simplifies initialization from the outside of all classes, since the controller will initialize the model and view within its own constructor, and the main class will only create an instance of the controller.

The main author of the Kivy library created an MVC template as well as an auto-generator for an MVC structure [35], [36]. These served as a great inspiration for the initial implementation of the FRED MVC. However, the view and controller would be difficult to separate in the event of a platform switch, since the view calls specific controller functions whenever an action is performed on the screen. This problem was remedied through dependency injection, which is the process of passing a reference to a function or object that is necessary to another class in order to eliminate any name dependencies. An example of this is the on_enter() function in each view, which is called upon switching into that screen. Each controller defines its own on_enter function and passes it to the view when initializing it, which gives the controller full discretion as to what is done in specific events. This example can be seen in Listing 4.1, where the ContactScreenController is passing its own definition of the on_enter_func(), which contains the functionality needed to initialize the screen to be ready for the user. The GenericScreenView is the class that all View classes inherit from. Since all screens will need a parsing function for speech recognition, an on_enter() function, and an on_leave() function, it makes sense to bake this functionality into a parent class in order to streamline implementation in child classes. As it can be seen in the ContactScreenView class, the only implementation needed in the constructor is dependency injections that are specific to the contact screen, such as the function for submitting a contact and the function for updating the contacts seen on the screen. This greatly speeds up implementation time and reduces coupling between the child view class and the controller class. If new functionality is desired from all screens in the program, this new dependency injection can simply be added to the GenericScreenView class. It also means that there will be less implementation

```python
class ContactScreenController(GenericController):
    def __init__(self, sm: SpeechMaster, db: MyPyrebase, oai: MyOpenAI):
        ...
        self._view: ContactScreenView = ContactScreenView(self.submit_contact, self.update_contacts, self.parse_func, self.on_enter_func, self.on_leave_func)
        ...

    ...

    def on_enter_func(self):
        self._model.populate_contacts()
        self._view.populate_contacts(self._model.contacts)

        # intro
        self._sm.makeSpeech(f"Let's make a contact together. Please say \
            their first name, last name, and number. You can also add their address", True)

    ...

class GenericScreenView(MDScreen, Subject):
    def __init__(self, parse_func, on_enter_func, on_leave_func, **kwargs):
        super(GenericScreenView, self).__init__(**kwargs)

        # initialize parse and enter funcs from those given by controller
        self._parse_func = parse_func
        self._on_enter_func = on_enter_func
        self._on_leave_func = on_leave_func

    ...

class ContactScreenView(GenericScreenView):
    def __init__(self, submit_contact, update_contacts, *args, **kwargs):
        super(ContactScreenView, self).__init__(*args, **kwargs)

        # initialize functions from controller
        self._submit_contact = submit_contact
        self._update_contacts = update_contacts

    ...
```

Listing 4.1: This code snippet showcases the GenericScreenView's advantages with default Kivy functions, as well as the dependency injection that the controller performs on the view.

mistakes, since the programmer does not have to rewrite any constructor logic for the base dependencies inside the child class whenever a new screen is implemented.

Each class, the model, view, and controller inherits from a generic class for each respectively. There are certain methods and properties that each class needs, no matter what the specific implementation is. For example, each view class needs an on_enter() function, which is the function called in Kivy when a screen is entered by the user. This inheritance follows the Open-Closed Principle, since the base class can be expanded to have more features in the case of an update to Kivy or more development in general. Additionally, new classes can easily be created to inherit from the base class in order to create more screens. This can significantly decrease the development time to create more screens and activities.

Each model in the program features Pythonic getters and setters which allow easier integration of data validation. In the Python standard, it is best practice to hide a class's data by using properties. These are indicated by an underscore at the beginning of the variable name, i.e. "_first_name." This allows the developer to indicate that these should not be accessed, rather their non-underscore counterparts. See Listing 4.2 for the syntax. This creates a simpler syntax for accessing the data from outside of the class, as well as building data validation into each access. When the developer accesses the non-underscored member, this automatically runs the getter or setter function seen in the figure. Developers of these classes can build whatever data validation into these functions.

It is important to acknowledge the downsides to this approach. First, code size is undoubtedly greater, since more abstraction is added to almost every step, such as retrieving a simple piece of information, like the contact's first name. This can create more overhead in maintaining, as well as in compute performance. This software design will also require longer for the uninitiated to understand and implement because of the previous downsides stated.

```
    def __init__(self, db_ref, **kwargs):
            super(ContactScreenModel, self).__init__(**kwargs)

            self._first_name: str = ""
        ...
    @property
        def first_name(self):
            '''first_name getter'''
            return self._first_name

    @first_name.setter
        def first_name(self, value: str):
            '''first_name setter'''
            if value is str and len(value) < 26:
                self._first_name = value
            else:
                raise ValueError("First names must be strings and < 
   26 characters")
        ...
```

Listing 4.2: Code snippet showcasing the getters and setters based on Pythonic properties, using the Contact screen as an example.

### 4.3.2 Observer-Subject

The list of observers is a simple container in the form of a Python list. The Subject adds new observers through its attach() function. When new information is ready to be shared, the Subject calls its notify_observers() function, which iterates through its list of observers and calls the update() function of each one. The Observer is a simple class that declares an abstract function called update(). The classes that inherit from the Observer, which is the controller of the screen, will define this function to specifically handle the update for its respective screen. For example, in the contacts screen, the update function extracts the first and last names, phone number, and address present in the text fields of the view. This is only called when the ”Create Contact” button is pressed.

### 4.3.3 Data Mapper

The data mapper implementation for this work takes on two different forms for different applications, but the Contact example can be seen in Figure 4.2. Notice that there are extra functions, exists() and read_all_contacts(), which are created to specifically be used within the contact screen. The main CRUD functions are always present, however, which allows an easy integration of database features. In Listing 4.3, the create function is a wrapper for database-specific code that abstracts away the implementation of this code from other instances that need to use it. The push_field() function receives a path list, much like os.path.join, which contains a list of strings representing each element in the path, and a value, which is whatever data needs to be pushed to the database. This value parameter needs to be formatted in the correct JSON format of the database. The correct format can be seen in the ContactMapper create() function, which receives a Contact as its only argument. The Contact is a simple class which only contains the simple strings that it needs, like first name and phone number. The contact_dict is a dictionary that can be sent to the database, since they are the same format as JSON. Lastly, the new_id string is the ID string of the newly created contact that is given by the database upon pushing this data. The ID string is a unique string made by the Realtime Database.

Since the data mapper pattern is so abstract in that it only needs to support CRUD, it can be used for other applications. For example, in this project, it is also used for CRUD operations on reminders. Since FRED is using the Google Calendar API, there is a need for an abstraction layer in order to be able to easily integrate other calendar software in the future, for example, Microsoft Outlook's calendar. Listing 4.4 shows the implementation of the create() function applied to reminders. It's obvious from this listing how cumbersome it would be to implement this code in other places. Additionally, it would be easy to integrate a new calendar API without the need for changing other code, like the controller for the reminder screen, since it simply calls this create function with the Event object. Lastly, the only new logic

```python
'''
=========================
In MyPyrebase...
=========================
'''
def push_field(self, path_list: list, value):
    # not logged in
    if self.user == None:
      return

    # iterate through children
    curr_field = self.db.child('users').child(self.user['localId']).
   child(path_list[0])
    for curr_child in path_list[1:]:
      curr_field = curr_field.child(curr_child)

    # try to get the value. errors out either b/c unauthorized or
   incorrect path
    try:
      val = curr_field.push(value, self.user['idToken'])
      return val['name']
    except HTTPError as e:
      print(e)
      return "NULL"

'''
=========================
In ContactMapper...
=========================
'''
def create(self, contact: Contact):
    contact_dict = {
        'first_name': contact.first_name,
        'last_name': contact.last_name,
        'phone_number': contact.phone_number,
        'address': contact.address
    }
    new_id: str = self._db.push_field(["Contacts"], contact_dict)

    # set id given by database push
    contact.db_id = new_id
```

Listing 4.3: Example of Create function for ContactMapper wrapping around a database-specific piece of code.

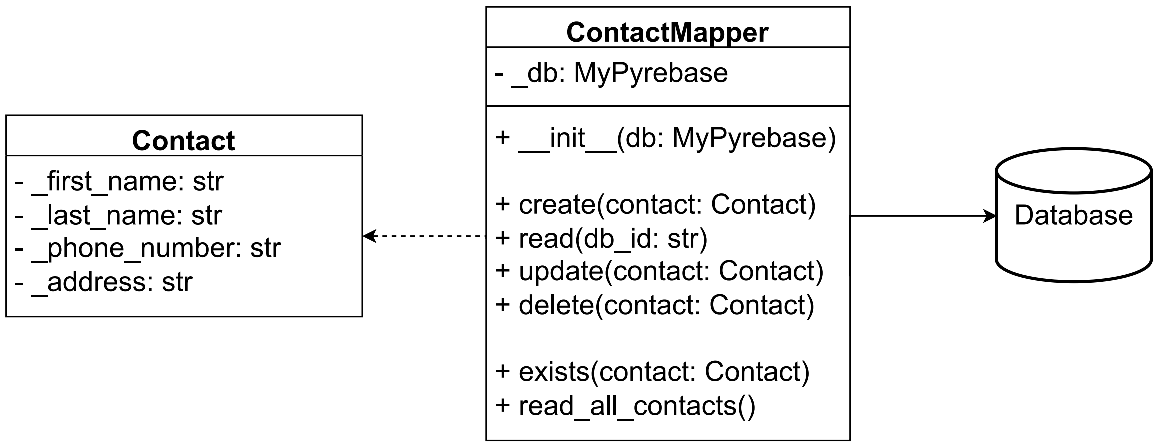


Figure 4.2: UML diagram of data mapper class. Example shown is for mapping Contacts.

```python
class EventMapper(object):
...

    def create(self, event: Event) -> Event:
        if self._service == None:
            return None

        calendar_event = None
        try:
            calendar_event = self._service.events().insert(
    calendarId='primary', body=event.to_dict()).execute()
        except HttpError as e:
            print(e)
            return None

        return calendar_event

...
```

Listing 4.4: Code snippet showing the ReminderMapper, which is another application of the data mapper pattern.

needed to create events of a different type would be in the Event.to_dict() function, where the Event class translates its python-specific datetime object into a dictionary that the calendar API can understand. This creates code that is easy to modify. The benefits of this abstraction is that the calling class that needs to create contacts does not have any understanding of how the database code is implemented.

## 4.4 Firebase

The information stored for each user needs to be backed up into an online database in order to make this data accessible to the user's caretaker. This information will only be accessible by the caretaker and will not be available for other users. This information includes settings, contacts, reminders, and responses to the check-in screen. The idea for future work is to create a companion app for FRED that the caretaker can install on their phone and view all of this data. An obvious choice is cloud storage solutions, because they are easy to integrate into an existing project, and do not require the upfront cost of maintaining a server. Firebase was the platform of choice for this, although other options are equally valid. Since users want to use their Google Calendar to save their reminders and events, it was easy to choose another Google-maintained service to link everything through one Google account.

The structure of the data is shown in Listing 4.5. Here, a branch called "users" denotes the section of data specific to users. Then, a long string is given as the user ID, which is created whenever a user makes their account. Within this branch, the user has fields like "Contacts," which will store all the information the user has for that subcategory. For example, each contact is pushed with an ID string which is pseudo-randomly created by Firebase itself. Inside of this branch is the specific fields pertaining to each contact, which is currently the address, first and last names, and phone number. Future work will add other fields for the settings and check-in results.

```json
{
"users": {
    "6A9Acv6U3meHTGtF9oOdnR2h1kl1": {
        "Contacts": {
            "-NNwAy-apJszjMM6dxtc": {
              "address": "123 main st",
              "first_name": "foo",
              "last_name": "bar",
              "phone_number": "1234567891"
            },
            ...
        }
        ...
    }
}
```

Listing 4.5: JSON snippet showcasing the database formatting inside the Firebase Realtime Database for user-stored contacts

## 4.5 Google Cloud Platform: Speech-to-Text and Text-to-Speech

Local STT and TTS are limited in their capabilities, especially on the Raspberry Pi. These small models are not performant enough for such a speech-heavy application. Therefore, it was decided to integrate a cloud-based solution. Again, since most of the online features of FRED were integrated in Google, an apparent choice was the Google Cloud Platform (GCP). In this solution, the Raspberry Pi streams byte arrays from the microphone using a Python generator. These pieces of audio information are processed by GCP, and the final speech result is decided when there has been a long enough amount of silence from the user. Additionally, the text-to-speech sounds very human-like, which is ideal for this application where seniors want more humanoid features in the robots.

The source code for the speech-to-text using GCP was adapted from the ”Perform streaming speech recognition on an audio stream” snippet from GCP's code samples. This code essentially creates a Python generator that acts as an audio stream for receiving byte arrays from the microphone of audio data and sending those incremental arrays to the GCP server. The highlights of the modified and enhanced source code can be seen in Listing 4.6. The original source code will continue to stream audio from the microphone after each sentence or phrase from the user until the program is stopped. However, for the FRED project, this is not ideal, since FRED will need to process what the user said, then provide feedback by saying something through TTS. If the audio stream keeps running, FRED will recognize itself when it is speaking, and an infinite loop will occur. To fix this, the audio stream checks if its "self.closed" flag is True, which means that a phrase or sentence has been found and processed. Additionally, after the processing has finished from the GCP server, and a phrase has been transcribed into text, the stream will set its own "self.closed" to True, which creates an automatic closing system based on the use case for FRED. Furthermore, since both the speech recognition and text-to-speech are threaded, a mutex is used in order to create a handshake between these two processes, ensuring that FRED will not be able to talk while it is listening for speech and vice-versa. This mutex is important, because some of the handshake involves checking flags that are set in other functions. The software needs to be certain that it waits to talk until the user is done talking, and so on. Although this can be done easily with synchronous code, this code all needs to be asynchronous, as the screen also needs to be active at all times to keep a responsive program. This is a fundamental aspect of mobile device programming, since a lot of the actions need to be asynchronous in order to allow the user and the system to do multiple things at once.

## 4.6 OpenAI

FRED needs the capability to create conversation with its users in order to create a relationship with the user and provide meaningful companionship. In order to do this, the power of GPT3 is used through the OpenAI API. It is worth noting that, at the time of implementing the system, GPT4's API had not yet been released for the public to use. Based on the research done with this model, the performance should

```python
class SpeechMaster(object):
    def halt_all_speech(self, exit_app: bool = False):
        # make sure we can acquire the lock
        self.recognition_lock.acquire()

        # stop speech recognition if we have active
        if self.stream is not None:
            self.stream.closed = True
        self.stop_speech = True
        self.exit_app = exit_app
        try:
            self.requests.close()
        except Exception as e:
            print(e)

        # stop the mixer if its playing
        if pygame.mixer.music.get_busy():
            pygame.mixer.music.stop()

        # release lock at end
        self.recognition_lock.release()

    def beginAllRecognition(self):
        self.recognition_lock.acquire()
        if (self.speech_thread != None and self.speech_thread.
    is_alive() and not self.exit_app):# or self.stopSpeech == True:
            print(f'recognition already started {self.speech_thread}
     {self.speech_thread.is_alive()} {self.stop_speech}')
            self.recognition_lock.release()
            return
        self.recognition_lock.release()

        self.speech_thread = threading.Thread(target=self.
    startSpeech)
        self.is_listening = True
        self.speech_thread.start()
        ...
```

Listing 4.6: This code snippet shows the highlights some of the implementation for the SpeechMaster class, which contains all of the code necessary for speech-to-text and text-to-speech

be much better. For this project, the task of interest is the automatic generation of conversation, also referred to as a "chatbot." FRED needs to be able to carry a casual conversation when the user is in the home screen, create unique reactions to user speech or interactions, and act as a mediator for speech-related games, like the story game. GPT3 is capable of doing all of these with impressive quality. The following is the prompt text used to start a chatbot that can conduct the story game:

```
The following is a conversation with a robot named FRED. The robot
   is helpful, creative, clever, and very friendly. The human will
   describe an image, and the robot respond to what the user said
   and will ask the user questions about the image.

FRED: I'd like you to tell me a story about the image shown on my
   screen. Say the words "I'm done" when you're finished.

Human: I see a tree and a mountain.

FRED: I love the outdoors. Do you like to go outside?

---

FRED: I'd like you to tell me a story about the image shown on my
   screen. Say the words "I'm done" when you're finished.

Human:
```

The beginning of the prompt is the introduction to the role that GPT3 is going to play during its text completions. It also tells the model what characteristics the chatbot will have, and can even give guidance on what each completion should look like. In this case, the model is told it is a robot named FRED, it has the characteristics of being helpful, creative, clever, and friendly, and it will respond to what a user says about an image and follow that response with a question about the scene described. Below this, an example conversation is given to further prime the model with how it should act during the session. Although the performance can be acceptable without being given an example conversation, some more complicated tasks require it in order to receive sufficiently intelligent completions. In this example, specifically, the model will not consistently give a response to what the user said, rather it will skip the response entirely and ask the user a question. This behavior is not desirable, because it does not foster a back and forth conversation between the user and FRED, rather it makes the user feel as though they are having a one-sided conversation.

Although the model is pre-trained, there are parameters that can be optimized to give better performance for the required task. The first is temperature, which controls how random the completions are that the model returns. The parameter ranges from 0-1, with 0 meaning that the model will become deterministic, always giving the same answer to a given prompt. For the story game, a temperature of 1.0 is used, since this gave the model more room to create different questions to ask the user. The next parameter is frequency penalty, ranging from -2 to 2, which penalizes tokens based on how many times they have occurred in the text already. Increasing this parameter will decrease the likelihood of the model giving the same completion multiple times. A value of 1.0 is used for the story game, since the model needs to ask the user many different types of questions to make the game more engaging. Another parameter is the presence penalty, ranging from -2 to 2, which applies a penalty to new tokens based on whether they exist in the text so far. Higher values will make the model more likely to cover new topics, and the value used in the story game is 1.0. There are other important parameters, such as Top P, but these are currently kept at their default values.

## 4.7 NeoPixel Coding

In the original prototype, the code to control the NeoPixel LEDs was handled by the Arduino in C. This code was very quick and simple. The problem with this solution is the fact that another device was necessary in order to run this feature.

This increases both the cost and complexity of the system, which leaves less room for other peripherals in the future. Therefore, it was decided to switch the LEDs to run natively on the Raspberry Pi through its GPIO pins. Unforunately, the Python NeoPixel library requires root access in order to execute on its preferred pin, GPIO18. This creates security concerns, because the main application should not be run with root access. The first intuition to solve this is to use a different pin. The SPI pin can theoretically be used for this, however, in practice, this pin caused undefined behavior where the LEDs would flash white off and on. The next idea to solve this was to create two programs that will communicate over a local socket. This was solution was also not secure, because a socket would be open that would allow communication from any process. The decided solution was to create two programs that separate the main application from the NeoPixel code. The NeoPixel code is run with root access, and a pipe is created between the two programs to communicate. The main program tells the LED code which face to display, including listening, talking or idle.

The layout of the NeoPixel disk is a simple series of rings in which the LEDs are indexed from 0-99, starting at the center LED. Each ring from there to the outer ring has an increasing multiple of 8 LEDs, so the second ring has 8, the third ring has 16, and so on. The code was designed to be modular in order to work on multiple sizes of NeoPixel disks. The Ring class was made to represent each layer of the disk, which contains a size and references to the LEDs that it represents. This class defines pattern functions which create an LED pattern based on a fill and accent color. For example, a simple fill() function will fill all of the LEDs in the ring with a certain color. Each pattern function creates a pattern for one time-step of the program. Each time-step is separated by an arbitrary number of seconds, for example 0.25 seconds. The number of seconds can be increased or decreased for different pattern behaviors. One can think of the patterns as the individual frames in an animation. The Disk class contains a list of Rings and definitions for characters that correspond to face patterns. For example, the character 0 represents the “listening” face.

## 4.8 Electrical Implementation

The original electrical design was created by FRED's 2021-2022 senior design team. The original electrical components included a Raspberry Pi 3B, an Arduino Nano, two speakers, an amplifier, the NeoPixel disk, a simple power switch and volume knob, a 5-inch LCD touch screen, and a 5V 8A power supply. The Raspberry Pi and Arduino Nano communicated with each other over a serial connection. In order to support the use of a Raspberry Pi 4 for a more powerful compute unit, a new wire was made to connect the Pi4 to the existing power solution, since the Pi3 uses a mini-USB connection, while the Pi4 uses a USB-C connection. Although the unit booted and was able to run the application, it did not work as intended, and would display a "Low Voltage Warning" while in use. Originally, it was thought that the increase in screen size to a 7-in screen or the increased LED changes in the NeoPixel ring may be the cause of the issue. However, through testing the current of these devices, there was still plenty of current available in the power supply. Thus, it was determined that the cause of the issue may simply be the new wire that was made. By creating another new USB-C cable with a sufficient wire gauge, the low voltage warning disappeared.

## 4.9 Testing: Focus Group

This section contains work that will be published in the Conference on Applied Human Factors and Ergonomics (AHFE) 2023 conference proceedings. This work discussed the FRED focus group performed with older adults with ADRD and their caretakers. I was the primary author of this work and contributed the software design. Dr. Xiaopeng Zhao and Dr. Kimberly Mitchell were the primary investigators. Luke MacDougall, Ella Hosse, and Matthew Rightsell were major contributors to the development of the FRED prototypes described here. Reproduced with permission from Springer Nature.

The study protocol outlined here was approved by the Institutional Review Board (IRB) of the University of Tennessee, Knoxville. The IRB number is UTK IRB-22-06870-XP. This study involved conducting a focus group with the target population of the FRED robot, including people with dementia and their caregivers, healthy older adults, nursing staff, and neurologists. Participants were recruited through physical flyers distributed by associated organizations or through emails. Participants were screened for eligibility for the study by conducting interviews via Zoom or phone. In order to determine their eligibility, participants were asked whether they had a legally authorized representative that signed documents for them. If so, they were asked if they could invite this person to the consent interview in order to have them review the consent process and determine their eligibility for the study. During the consent interview, the participant was briefed on the study procedure and the consent forms approved by the IRB. In order to determine their ability to consent, the Decision-Making Capacity Assessment Tool (DMCAT) was administered to the participant. If they were unable to complete this test, the participant was asked if they have a legally authorized representative that can consent on their behalf. If not, the participant was thanked for their time and was not included in the study. No data was collected on any representative of the potential participant during this consent interview.

At the beginning of the focus group, each participant signed a new consent form and completed a preliminary survey that asked demographic questions, as well as their thoughts and feelings about dementia care if they were a caretaker for a person living with ADRD. If the participant consented to the study, they were assigned name tags with "PX" written on them, where X is an identifying number to know how their answers relate to the demographic survey taken, however, there was no record taken of matching names to participant identifiers at any point during the study. Afterwards, the question and answer session was conducted, where they were asked questions that relate to dementia care and living with dementia. After the session, there was a lunch break in which no notes were taken. Finally, after the break, participants were divided into three groups: two groups that viewed the two different interface designs on tablets and one group that interacted with the FRED robot itself. During both sessions, notes were taken by several researchers on their answers to questions and feedback on the interfaces and robot.

During the question and answer session, participants identified some important needs for the user interface (UI) design. One important aspect that was agreed upon by all was the fact that the interface design needs to stay the same once a person with ADRD begins using it. Participants did not like the idea that the UI could be updated at a later point, and applications or activities would be arranged differently from how they were before. Another important aspect of the disease that was repeated among participants is the need for routine. Additionally, they identified that they want the ability to personalize the layout and available features within the app to their individual needs. Lastly, when asked if machine learning or artificial intelligence would be unsettling if implemented in FRED, many participants were open to the idea as long as they had assurance that their data remained private.

During the interactive session, participants identified a few aspects of the current design that they do not prefer. First, participants did not like the face LEDs, with multiple participants stating that it would give them a migraine, and that it would be too distracting. One participant stated that a new experience from the disease is that they are very easily distracted, and that this constant LED movement would be too much to handle. Also, participants said that the contacts on the contact page were too small for them. Lastly, participants did not like the idea of having multiple levels of interface pages, meaning that if they have to navigate through an intermediary screen to get to the next activity, this would be confusing for them to navigate, especially when they would need to navigate backwards. They would prefer to have one central screen that they always return to, such as the home screen. They also identified parts of the current design that they liked. Participants were excited by the interactive conversation of FRED, and said that they would like to have even more of that. They also thought that the check-in page was helpful for caregivers.

# Chapter 5

# Conclusion and Future Work

## 5.1 Conclusion

In this work, an affordable robot system for people living with ADRD was designed and implemented. The first prototype was a rudimentary system based on an Android phone and an Arduino communicating over bluetooth. The system evolved from this preliminary state into a system that is based on a Raspberry Pi and peripherals, a 3D-printed body, and a custom graphical user interface. To enable interaction between the user and FRED, multiple modes of communication were implemented, including touch, speech-to-text, and text-to-speech. In addition, the power of GPT-3 was applied to FRED's interaction system to create a rich user experience in which responses from FRED are not merely scripted responses. To support this richness in experience, FRED was also equipped with a full NeoPixel disk face, which has 100 controllable LEDs to give the impression of reactions and states of being from FRED. The system was tested with multiple groups of potential users to evaluate the efficacy and robustness of the system. The feedback gathered from these test groups helped guide the development of the FRED system in a meaningful way. Principles and patterns of software development were employed throughout the programming of the system to ensure the longevity of the system as a development and research platform, including the model view controller and data mapper patterns. Additionally, since affordable hardware and easy-to-install programming platforms were used to create the system, it is easily reproducible for conducting more research on these affordable social robots.

## 5.2 Future Work

Future work should seek to develop and finish more user-requested features, like a fully-fledged reminder system and the ability to make calls. Although the system has a sufficient framework for more complex features, the current features are not yet ready to fulfill user needs to a sufficient extent. For example, the reminders are able to be made an pushed to the user's Google Calendar, however, the sign-in system is not user-friendly for authenticating with Google, and users are not yet able to delete or modify existing reminders through the current GUI. Additionally, the current system takes some technical expertise to install the required dependencies. This could be a roadblock for other researchers, as not all those in the space will be experienced with Raspberry Pis or Linux-based operating systems. As such, future work should seek to create an installer or a simple script for installing all the required dependencies and Python packages for Raspian. Lastly, more work should be done to refine the OpenAI features. At the time of developing these features, GPT-3 was the most capable model, but with the release of GPT-4, the AI capabilities of FRED could be further expanded. The implementation and testing of the GPT-4 model should be carried out for an even more capable system.

# Bibliography

[1] Alzheimer's Association, "2022 Alzheimer's disease facts and figures," *Alzheimers Dement*, vol. 18, no. 4, pp. 700–789, 2022. DOI: 10.1002/alz.12638 (p. 1).

[2] M. Miller, "The future of u.s. caregiving: High demand, scarce workers," *Reuters*, 2017. [Online]. Available: https://www.reuters.com/article/us-column-miller-caregivers/the-future-of-u-s-caregiving-high-demand-scarce-workers-idUSKBN1AJ1JQ (p. 1).

[3] S.-J. Hong, E. Ko, M. Choi, N.-J. Sung, and M.-I. Han, "Depression, anxiety and associated factors in family caregivers of people with dementia," *Journal of Korean Neuropsychiatric Association*, vol. 61, no. 3, p. 162, 2022. DOI: 10.4306/jknpa.2022.61.3.162 (p. 1).

[4] F. Yuan, E. Klavon, Z. Liu, R. P. Lopez, and X. Zhao, "A systematic review of robotic rehabilitation for cognitive training," *Frontiers in Robotics and AI*, vol. 8, p. 105, 2021 (p. 1).

[5] J. Hirt, N. Ballhausen, A. Hering, M. Kliegel, T. Beer, and G. Meyer, "Social robot interventions for people with dementia: A systematic review on effects and quality of reporting," *Journal of Alzheimer's Disease*, vol. 79, no. 2, pp. 773–792, 2021. DOI: 10.3233/JAD-200347 (p. 1).

[6] J. Gerłowska, M. Furtak-Niczyporuk, and K. Rejdak, "Robotic assistance for people with dementia: A viable option for the future?" *Expert Review of Medical*

*Devices*, vol. 17, no. 6, pp. 507–518, 2020. DOI: 10.1080/17434440.2020.1770592 (p. 1).

[7] S. Robotics, *Pepper the humanoid and programmable robot*, Minato City, Tokyo, Japan: Aldebaran, 2022. [Online]. Available: https://www.aldebaran.com/en/pepper (visited on 05/01/2023) (p. 1).

[8] S. Robotics, *Nao the humanoid and programmable robot*, Minato City, Tokyo, Japan: Aldebaran, 2022. [Online]. Available: https://www.aldebaran.com/en/nao (visited on 05/01/2023) (p. 1).

[9] R Sather, M Soufineyestani, A Khan, and N Imtiaz, "Use of humanoid robot in dementia care: A literature review," *J Aging Sci*, vol. 9, p. 249, 2021 (p. 1).

[10] P. R. USA, *Paro therapeutic robot*, Itasca, Illinois, United States: PARO Robots USA, 2014. [Online]. Available: http://www.parorobots.com/index.asp (visited on 02/15/2023) (pp. 1, 2).

[11] Sony, *Aibo: Robotic puppy, powered by ai*, Minato City, Tokyo, Japan: Sony, 2023. [Online]. Available: https://us.aibo.com/ (visited on 05/01/2023) (p. 1).

[12] L. Pu, W. Moyle, C. Jones, and M. Todorovic, "The effectiveness of social robots for older adults: A systematic review and meta-analysis of randomized controlled studies," *The Gerontologist*, vol. 59, no. 1, e37–e51, 2019. DOI: 10.1093/geront/gny046 (p. 2).

[13] H. S. Kang, K. Makimoto, R. Konno, and I. S. Koh, "Review of outcome measures in paro robot intervention studies for dementia care," *Geriatric Nursing*, vol. 41, no. 3, 207–214, 2020. DOI: 10.1016/j.gerinurse.2019.09.003 (p. 2).

[14] R. C.-S. Chang, H.-P. Lu, and P. Yang, "Stereotypes or golden rules? exploring likable voice traits of social robots as active aging companions for tech-savvy baby boomers in taiwan," *Computers in Human Behavior*, vol. 84, pp. 194–210,

2018, ISSN: 0747-5632. DOI: https://doi.org/10.1016/j.chb.2018.02.025. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0747563218300839 (p. 3).

[15] J. Laski, "Intuition robotics launches elliq, the award-winning care companion robot, for commercial sale," *PR Newswire*, 2022. [Online]. Available: https://www.prnewswire.com/news-releases/intuition-robotics-launches-elliq-the-award-winning-care-companion-robot-for-commercial-sale-301502682.html (visited on 02/15/2023) (p. 3).

[16] R. C. Martin, "Design principles and design patterns," *Object Mentor*, vol. 1, no. 34, p. 597, 2000 (p. 3).

[17] B. Meyer, *Object-oriented Software Construction* (Object-oriented programming). Prentice Hall PTR, 1997, ISBN: 9780136291558. [Online]. Available: https://books.google.com/books?id=xls_AQAAIAAJ (p. 4).

[18] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, "Design patterns: Abstraction and reuse of object-oriented design," in *ECOOP' 93 — Object-Oriented Programming*, O. M. Nierstrasz, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 1993, pp. 406–431, ISBN: 978-3-540-47910-9 (p. 5).

[19] G. E. Krasner, S. T. Pope, *et al.*, "A description of the model-view-controller user interface paradigm in the smalltalk-80 system," *Journal of object oriented programming*, vol. 1, no. 3, pp. 26–49, 1988 (pp. 5, 33, 34).

[20] M. Fowler, *Patterns of enterprise application architecture.* Addison-Wesley, 2015 (p. 6).

[21] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, *Attention is all you need*, 2017. arXiv: 1706.03762 [cs.CL] (p. 6).

[22] T. B. Brown *et al.*, *Language models are few-shot learners*, 2020. arXiv: 2005.14165 [cs.CL] (p. 7).

[23] OpenAI, *Gpt-4 technical report*, version 3, 2023. arXiv: 2303.08774 [cs.CL] (p. 8).

[24] P. Christiano, J. Leike, T. B. Brown, M. Martic, S. Legg, and D. Amodei, *Deep reinforcement learning from human preferences*, 2023. arXiv: 1706.03741 [stat.ML] (p. 8).

[25] N. Staudacher, *What is chatgpt?* OpenAI, 2023. [Online]. Available: https://help.openai.com/en/articles/6783457-what-is-chatgpt (p. 8).

[26] R. Bray, L. MacDougall, C. Blankenship, *et al.*, "Development and assessment of a friendly robot to ease dementia," in *Social Robotics*, vol. 13818, Cham: Springer Nature Switzerland, 2022, pp. 381–391, ISBN: 978-3-031-24670-8 (p. 10).

[27] Arduino, *Arduino uno wifi rev2*, New York, New York, United States: Arduino Store, 2021. [Online]. Available: https://store-usa.arduino.cc/products/arduino-uno-wifi-rev2?selectedStore=us (visited on 03/12/2022) (p. 14).

[28] Adafruit, *Raspberry pi 4 model b - 1 gb ram*, New York, New York, United States: Adafruit, 2023. [Online]. Available: https://www.adafruit.com/product/4295 (visited on 03/12/2022) (p. 14).

[29] Arduino, *Arduino nano*, New York, New York, United States: Arduino, 2021. [Online]. Available: https://store-usa.arduino.cc/products/arduino-nano?selectedStore=us (visited on 03/12/2022) (p. 14).

[30] M. Technology, *Atmega4808/4809 data sheet #40002173*, Chandler, Arizona, United States: Microchip Technology, 2023. [Online]. Available: https://www.microchip.com/en-us/product/ATMEGA4809 (visited on 03/12/2022) (p. 16).

[31] Arduino, *Getting started with arduino*, New York, New York, United States: Arduino, 2023. [Online]. Available: https://docs.arduino.cc/learn/starting-guide/getting-started-arduino (visited on 03/12/2022) (p. 17).

[32] D. Barto, C. W. Bird, D. A. Hamilton, and B. C. Fink, “The simple video coder: A free tool for efficiently coding social video data,” *Behavior Research Methods*, vol. 49, no. 4, 1563–1568, 2016. DOI: 10.3758/s13428-016-0787-0 (p. 22).

[33] I. Aizenberg, *Emteria for raspberry pi 4 model b is now available as download*, Aachen, Nordrhein-Westfalen, Germany: Emteria, 2022. [Online]. Available: https://emteria.com/blog/emteria-os-for-raspberry-pi-4b (p. 27).

[34] R. Pi, *Raspberry pi 4 model b product brief*, Cambridge, United Kingdom: Raspberry Pi Foundation, 2021. [Online]. Available: https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications/ (visited on 03/12/2022) (p. 41).

[35] HeaTTheatR, *Kivy mvc template*, 2021. [Online]. Available: https://github.com/HeaTTheatR/Kivy_MVC_Template (visited on 04/05/2023) (p. 45).

[36] A. Rodríguez, I. Yuri, A. Bulgakov, *et al.*, *Script creates a project with the mvc pattern*, KivyMD, 2022. [Online]. Available: https://kivymd.readthedocs.io/en/1.1.1/api/kivymd/tools/patterns/create_project/index.html (visited on 04/05/2023) (p. 45).

# Appendix

# Appendix

## A Prototype 1: Usability Study

Table 1: Demographics of participants (n=16). Participants were given the following options for the familiarity with technology questions: 1=Not at all familiar, 2=Slightly familiar, 3=Moderately familiar, 4=Very familiar, and 5=Extremely familiar.

| Demographic variable | Statistic |
|---|---|
| *Age* | *n* (%) |
| 18–30 | 16 (100%) |
| *Gender* | *n* (%) |
| Female | 5 (31%) |
| Male | 11 (69%) |
| *Highest level of education* | *n* (%) |
| 7th-11th grade | 2 (13%) |
| High school graduate | 2 (13%) |
| Some college | 9 (56%) |
| College graduate | 2 (13%) |
| Post-graduate | 1 (6%) |
| *Familiarity with technology* | M $\pm$ SD |
| Smartphones | 4.06 $\pm$ 1.06 |
| Tablets | 3.88 $\pm$ 1.088 |
| Computers | 4.06 $\pm$ 1.063 |
| Robots | 2.93 $\pm$ 0.704 |

# Vita

Robert Bray grew up in Savannah, Georgia. He attended high school in Hummelstown, Pennsylvania. After high school, he attended the University of Tennessee, Knoxville and received a Bachelor of Science degree in Computer Science. After graduating, he chose to attend the University of Tennessee, Knoxville for a Master of Science degree in Computer Science. His research interests include software design principles and machine learning. After graduating, he will pursue a career in software engineering and machine learning. He is overwhelmingly grateful for the support from his family and friends throughout his education and as he begins his new career.