To the Graduate Council:

I am submitting herewith a thesis written by Bryceton Bible entitled "EASIER AIR ALERT PLATFORM: A DESIGN AND APPROACH TO CREATING A DISTRIBUTED AIR QUALITY MONITORING AND ALERT SYSTEM." 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. Audris Mockus

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 Bryceton Bible entitled "EASIER AIR ALERT PLATFORM: A DESIGN AND APPROACH TO CREATING A DISTRIBUTED AIR QUALITY MONITORING AND ALERT SYSTEM." 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. Audris Mockus

Accepted for the Council:

Dixie L. Thompson

Vice Provost and Dean of the Graduate School

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

EASIER AIR ALERT PLATFORM: A DESIGN AND APPROACH TO CREATING A DISTRIBUTED AIR QUALITY MONITORING AND ALERT SYSTEM

A Thesis Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

Bryceton Bible

May 2024

Dedicated to my parents for always supporting me and creating my love of learning, to Dr. Zhao and Horiguchi-sensei for countless amazing opportunities. Thank you forever to Korone for smiles and laughter along the way.

Acknowledgements

The Tennessee Valley Authority has partnered with local research organisations to encourage development of systems that benefit Tennessee communities. On their website, TVA describes that they are "funding solutions to equitably improve our communities in the Valley through innovative solutions with broad-range opportunities and applicability". TVA [2024] The specifics of these projects vary, but fall into categories such as "Economic Empowerment", "Broadband and Digital Literacy", "Energy & Environmental Justice", and "Enhanced Community Resiliency." The focus of this paper is the so-called "EASIER project" (Elders Alert System for Imminent Environmental Risk), which was accepted after its pilot was submitted to the "Energy & Environmental Justice" category by the Three3 research nonprofit under the title "Alerting Elders to Air Quality Health Risks." The project "pitch" as described on TVA's list of pilots summarises its relationship to Environmental Justice succinctly: "To improve the health and wellness of under-resourced elders, Three3, Inc. is implementing strategies to improve resiliency to indoor and outdoor environmental health risks." From this pitch, there are obvious requirements but also significant flexibility in regards to development specifics and actual implementation of such a system. The EASIER project receives funding from this TVA grant, via the nonprofit Three3. The scope of this paper covers the technical development of the software and integration of associated hardware, which was (and is) performed by Dr. Zhao's lab via a contract with Three3, Inc.

絶えず君のいこふ　記憶に夏野の石一つ

Taezu kimi no ikou, kioku ni natsu no no ishi hitotsu

Eternally, you are in my memory, a single stone in a summer field

Abstract

This thesis presents the design approach, development and implementation of the Elders Alert System for Imminent Environmental Risk (EASIER) project, an air quality monitoring and alert system aiming to improve the health and wellness of under-served elder communities, as a part of the Tennessee Valley Authority Connected Communities initiative for Environmental Justice. The EASIER project provides homes with a fully integrated, connected system capable of real-time air quality monitoring, notifications and descriptions of potential air quality risks, and educational material to empower these community members to take charge of their own air health. Further, EASIER aims to inform relevant family/friends of identified risks automatically when they are detected. The EASIER system utilises off-the-shelf components, including Android tablets, AWAIR air quality monitoring systems, backup wireless access points and batteries, and a central web server to connect devices and process their data. This thesis describes the project's goals, its initial design requirements, the overall system architecture, its implementation, and project results so far, providing an overview of a scalable architecture capable of educating and enhancing the environmental health of vulnerable populations.

Table of Contents

Chapter 1

Introduction

In this section, the background and preliminary information regarding this project will be introduced.

1.1 Project Goals

The focus of this paper is the so-called "EASIER project" (Elders Alert System for Imminent Environmental Risk), which was accepted after its pilot was submitted to the "Energy & Environmental Justice" category by the Three3 research nonprofit under the title "Alerting Elders to Air Quality Health Risks." The project "pitch" as described on TVA's list of pilots summarises its relationship to Environmental Justice succinctly: "To improve the health and wellness of under-resourced elders, Three3, Inc. is implementing strategies to improve resiliency to indoor and outdoor environmental health risks." From this pitch, there are obvious requirements but also significant flexibility in regards to development specifics and actual implementation of such a system. The scope of this paper covers the technical development of the software and integration of associated hardware, informed by this community's heightened need for more accessible and understandable health communication (Weiss et al. [1995]).

1.2 Initial Design Requirements

This section describes the goals of the EASIER project. This will be done heuristically, in a few stages with different focuses. Goals of the project as a whole will be described, predetermined acquired hardware will be described, and a summary of Three3 provided design requirements will covered in this section.

1.2.1 Goals

The goal of the EASIER project is to furnish each participating home with a fully integrated, connected system to allow measuring of their home's air quality. In doing so, the system should notify the users of potential air quality risks, provide information regarding potential causes and possible ways to mitigate these risks, and when necessary contact friends/family/members of the community to notify unaddressed risks on the individual's behalf. From these broad goals (in combination with already acquired hardware and services) implementation specifications were created at the onset of development, which will be discussed thoroughly in later sections. The development of Initial Design Requirements did mandate the selection of certain tools/frameworks/APIs, as this project's components are often interdependent. Where relevant, this section will discuss this implementation's specifically selected options. However, the modular design of EASIER would allow most of these components to be interchanged without issue.

1.2.2 Hardware and Services

After the pilot's acceptance by TVA led to initial funding, prerequisite hardware and services were acquired for the project. This hardware led to many of the original software design requirements for the project, though even with hindsight most decisions would be repeated. Decisions in regards to hardware and service selection were made by Three3, and were set in stone before the scope of this paper.

However, as these decisions were integral to the implementation of the project and mandated certain approaches for software creation, they will be briefly covered here in the "Initial Design Requirements" section, as they have remained mostly unchanged during the duration of this project. Each home's system includes the following:

Provided hardware:

- A Samsung tablet running Android
- An AWAIR Element air quality monitoring system
- A mobile hotspot
- A backup battery power supply

Provided services:

- Mobile internet connectivity
- Access to Google services
- Connectivity to the AWAIR professional platform

Each Samsung tablet was equipped with a protective case and screen protector. Participants are expected to interact regularly with their tablet, often for purposes outside of this project. To ensure longevity and prevent damage to the tablets, the tablets are provided to the homes already in these protective cases. Each tablet runs a recent version of Android and has access to the Google Play Store and other Google services. A mobile hotspot with an active SIM is included with each system. The tablets access the internet through this hotspot, as it is not a guarantee that every participating home (especially in this demographic (Choi and DiNitto [2013])) will have their own stable wireless internet access point. Lastly, out of desire to ensure system stability, all units are connected to a battery backup power supply, allowing the system to continue operating even when mains power is unavailable.

1.2.3 Three3 Design Requirements

Some broad design requirements were specified by Three3 at the onset of this project. These requirements described desired features and outcomes for the project.

- The system should be accessible from the tablets.
- The system should be able to communicate with the AWAIR Element air quality monitoring system to receive air quality information.
- The system should relay this information to the user in an easy to understand way.
- The system should generate alerts when the measured air quality information violates a set of "rules" indicating a potential health risk.
- These alerts should be provided to the user as quickly as possible.
- These alerts should also be able to be provided to opted-in social network contacts (who do not have direct access to the system) by some means.

These requirements leave ample room for flexibility in implementation. A keen reader might understand this flexibility from the wording of the first design requirement: "the system should be accessible from the tablets", which itself does not even explicitly define the obvious implementation choice of an Android application, or the final requirement's lack of inclusion of a specific communication channel. The great flexibility in requirements of this project allowed exploration of new and interesting approaches to this full-stack system implementation, but also led to a fair amount of changes made to the implementation as it progressed.

1.3 Definitions and Terms

Before continuing to more specific content of this paper, there are some frequently used, project-specific terms that must be understood. This section presents definitions of terms that have meanings specific to this project that may be ambiguous otherwise.

- **Alert Rule**: A document defining a number of air quality "alert rules" was created for this project. These alert rules describe things like upper and lower temperature thresholds, pollution thresholds, and other relevant factors. These alert rules were designed by collaborating air quality experts. An alert rule is one specific "threshold" that may generate an alert.

- **Alert**: An alert is generated when a given alert rule is violated. For example, if an alert rule exists for exceeding a temperature of 90 degrees, a single alert would be generated if a temperature of 91 degrees is measured. From a technical perspective, in this project description, an alert may refer to this conceptual alert, or to the actual implemented code-based "object", or potentially a database entry created when an alert is generated.

- **System**: A system (or "unit") is the fully integrated tablet/sensor/backup package that is installed in participating homes.

- **Social Network**: Participants in the EASIER project are encouraged to specify opted-in contacts (usually family or friends) that will receive high-risk alerts automatically.

- **AQI**: Air Quality Information. Temperature, humidity, pollutant concentrations, etc.

- **Push Alert**: A notification (usually containing information generated with an alert) sent to the Samsung tablet as a "push notification" via the Google Cloud Messaging Service.

- **In-app Alert**: A notification (usually containing information generated with an alert) sent to the Samsung tablet via our own API.
- **SMS Alert**: A notification (usually containing information generated with an alert) sent to a user's social network via the Twilio SMS API.
- **EASIER Web Server (EWS)**: The EASIER Web Server, as defined in the General Architecture (2.2.1) may be referred to as *EWS* or *EASIER Central Server* throughout this paper.
- **EASIER Homes Panel**: The EASIER Homes Panel is a web based application developed to run on the EASIER Web Server to allow team members to interact with the server. Specifically, features like adding new sensor IDs and other home information to the database, editing existing entries, adjusting Alert Rule thresholds, viewing recorded data, and performing test actions like API calls that are typically automated. Most of the Homes Panel was designed to be used by non-technical team members, so creating an easy to understand, responsive user interface was critical.

1.4 Specifications

1.4.1 Software Development Platform

As the primary means of user interaction for this project is via the Samsung tablets, development of an Android application was deemed ideal. Historically, Android application development has been rather cumbersome and limiting. Further, in the case that the application needed to be made available on other devices (like in the case that a different type of tablet was chosen for future systems), traditionally there would be a need for work to be completed in order to create executables for other platforms. For this reason, Flutter was selected as the primary software development platform for the EASIER mobile application.

Flutter is a software development kit created by Google. Google describes Flutter as their "UI toolkit for building beautiful, natively compiled applications for mobile, web, and desktop from a single codebase" (Google [2024]). In short, Flutter allows one codebase to compile to applications for multiple platforms with consistent UI. Compiling an application to a different platform typically requires little or no modification (with the exception of specific device-dependent tools). With Flutter, application builds work out of the box on not only the targeted Samsung tablet with its large screen size, but typical Android phones, allowing easier testing and onboarding of new team members. Additionally, the application can be tested on a Windows (or Mac or Linux) PC without need for cumbersome emulation software. Lastly, should the team decide to provide non-Android tablets (like iPads) to homes, minimal modifications to the application's source code would enable all necessary functionality for the EASIER system. Flutter projects are primarily coded in the Dart programming language. Dart, also developed by Google, is object-oriented and class-based, and has syntax that is familiar to most experienced programmers.

1.4.2 Software Distribution Platform

Since many of the users are unfamiliar with operating an Android device, it was determined that there is a need to implement a system to automatically install application updates to users' devices. Therefore, distribution of the EASIER application is conducted via the Google Play Store and its associated services. To enable distribution and updating via the Google Play Store, a Google Play developer account was created for the project. Via this account, the application is made available on the Google Play store publicly. This required Google approval, which was quickly granted. While it is perhaps a security risk to expose the application to the general public, the application is of little use to any who might come across the application without device API keys which are manually entered in the application settings after install.

1.4.3 Remote Web Server

Rather than requiring each Android device to persistently send HTTP requests to AWAIR's servers for fetching measurements from air quality sensors, a central web server (referred to in this paper as the EASIER Central Server, EASIER Web Server or EASIER Remote Web Server) was created to function as an intermediary. This was found beneficial for multiple reasons: hard-coding Alert Rules into the Flutter source code is not ideal as Alert Rules may need to be adjusted or added/removed at any time. Further, if Alert Rules were implemented on a remote web server and each device polled the AWAIR server for sensor data on its own, each device would need to fetch its sensor data, send it to the EASIER server to determine if an alert needed to be generated, and receive said alert back from the server (via some sort of REST-API or similar). This would be a very complex implementation that would rely on the application being active so each device could send multiple requests every few seconds to both our and AWAIR's server. With these factors in mind, the EASIER Remote Web Server was created. The EASIER Server handles all communication with AWAIR's API (as well as the notification APIs) and processing of alerts and alert rules. Every few minutes, the server requests air quality information across *all* devices. The server then generates alerts according to alert rules stored on the server (allowing rules to be changed dynamically), and sends Push/SMS notifications when applicable. Using this approach, all collected sensor data is centrally available, Alert Rules can be changed and examined instantaneously, and updates to alert and API logic can be changed independent of the Android devices. Data collection, alert generation, and notifications are carried out regardless of Android device activity/connectivity, enabling full functionality even when users are away from the application. When the application is open, the Flutter application can contact the EASIER Server over HTTP to request any generated alerts and other information that needs to be displayed. This asynchronous approach scaleably manages communications between multiple servers, making implementation of notifications and per-home alert rules

straightforward. For the Remote Web Server implementation, a simple Apache based commercial shared web hosting platform is used. Core feature requirements include:

- The ability to create and interact with databases. EASIER utilised MySQL databases.
- Server-side scripting . EASIER relies mostly on PHP.
- The ability to respond to and send HTTP requests of some sort. Apache and PHP support GET/POST style requests.
- Automation support. EASIER uses Cron Jobs to accomplish automation.

1.4.4 Device Messaging Platform

When the EASIER application is open, frequent HTTP requests are sent to the EASIER server, which allows in-app notifications to be implemented simply: if the HTTP request returns an alert, then logic in Flutter opens a popup window to convey the necessary information. This technique is dependent on the application being active to send HTTP requests. However, most users will not constantly have the application open and active for many reasons. Thus, some sort of non-application dependent push notification system (whose effectivity is known (Adobe [2024])) is needed. While Android applications can themselves create push notifications on a user's device, typically a web-based external service is required to create push notifications when a user is not actively using a given application. Google's primary solution to over-the-air push notifications is the Firebase based Cloud Messaging service (FCM). Through just a few lines of code, a background service is created on the Android device that checks continuously for undelivered messages. Each device reports its unique FCM ID to the EASIER Server, which is used to trigger targeted push notifications containing alert contents.

1.4.5 Community Messaging Platform

To fulfill one of the main requirements of this project, a means for sending notifications of alerts to social network contacts must be implemented. Given that these social network contacts do not have direct access to the user's device and cannot be expected to install an application, communication channels are limited. Further, it is beneficial to send the alert notifications via a channel that will be immediately received (as opposed to something easily ignored like email) since these alert notifications are often indicative of an urgent issue. Thus, a system for sending alert notifications via SMS text message is ideal. Messages only need to provide a few words social network contacts to encourage checking in on potentially at risk users. Since it can be assumed that all opt-in social network contacts have a reachable phone number, this approach does not impose significant burden on social network contacts. Twilio was selected to be the SMS platform due to its ease of use and accessible price, costing only a few cents per message. In addition to enabling social network notification via SMS, preliminary discussions are in place to create further "community contacts" for severely urgent alerts. For example, if a home's temperature reaches extremely unsafe levels, it may become necessary to contact local health authorities, fire departments, police, etc - to perform a welfare check on the home.

1.4.6 Weather Platform

To generate more meaningful alerts and advice for participants, accessing local weather information and predictions was identified as a useful feature for both the user-facing application and server-side features. Open-Meteo was selected for its ease of use. Weather information is presented to users on the homescreen of the application.

Chapter 2

General System Architecture

In this section, the system's general architecture will be described. While the preceding section lays out specific design and component choices that were made based on provided project requirements, this general system architecture will describe broadly a platform-independent architecture for this kind of monitoring and alerting system. For this General System Architecture, the interaction between components and their individual purposes are explored, even though they could be implemented in many different ways. These components interact with each other using a predetermined method of communication, but could be implemented many different ways. Microsoft's summary of the concept of a Service Orientated Architecture using solely the word **agility** serves to demonstrate the flexibility of an actual implementation of the EASIER system. Microsoft describes such a type of construction as one which uses well-defined message exchanges to communicate between distinct services that are loosely coupled, allowing reuse of existing systems wrapped into modules that can be "plugged and played" as needed (Microsoft [2016]). This Service Orientated architectural approach enables projects to more easily and quickly implement features, via re-usability and the potential to swap components/services as needed (Mankad and Sajja [2010]).

Later, details regarding tangible interactions between actual implemented system components will be described more specifically

2.1 What Are Alert Rules and Alerts?

As Alerts and Alert Rules will be referenced many times in the description of EASIER's architecture, it is important to understand their respective purposes and uses. An Alert is generated when a given Alert Rule is triggered. Alert Rules have an associated threshold, a sensor type, an operator, and in some cases more complex considerations. An example of an Alert Rule might look like: $CO2 < 500$. These alert rules were developed in collaboration with air quality experts that are involved in the project and its development. Alert Rules might apply to all units in the EASIER system, or might apply to only specific home(s). The result of an Alert Rule triggering is called an "Alert". For example, given the above Alert Rule example, if CO2 is measured to be 600, an Alert would be triggered/created. In the current EASIER implementation, this generated Alert contains information like the sensor reading that led to the violation, timing information, severity information, home ID, etc. These Alert objects are generated on the server and stored in a database.

2.2 System Architecture

This section describes the highest possible view of the architecture of the EASIER project without going into detail about actual implementation. A representation of the most general description of this project's architecture can be found in Figure 2.1. This chart is designed using a form of UML modified for understandability. Unified Modeling Language (UML) is a framework for visualising and modeling architecture of and interactions of systems, especially in (but not limited to) the field of engineering (Thomas [1997]). UML is also used in describing software architecture (Selic [2008]) and is capable of describing highly complex systems. However, due to

the relative simplicity of this project, a simplified pseudo-UML format will be used for architectural visualisations throughout this paper.

2.2.1 Architecture Description

Components

In the EASIER system, there are six main components:

- **User Home**: The user's home is furnished with necessary sensors and devices, as well as accounts and connectivity to allow intercommunication between devices and components. Sensors, power supplies, wireless access points, and User Device/tablets are assigned to a specific home. A home is considered its own component because it is independent of the User Device (tablet) in that data collected from sensors is associated with a home, not a specific User Device (tablet) or set of sensors. Devices, including sensors, could be replaced or interchanged without affecting the system as a whole.

- **User Device**: The device provided to the user to interface with the EASIER system. Per system requirements, this device must be capable of connecting to the internet and be able to run a mobile application.

- **EASIER Web Server**: A central server used for distributed collection of sensor data, external data from other components, communicating with messaging APIs, logging data, communicating with user devices, and handling/storing Alert Rules. Communication to/from the EASIER Web Server and other components is via HTTP requests. Further, the central server must be capable of reading and writing to a database. This is a core feature of most web server platforms. Web server platform and database platform selection are codependent.

- **Cloud Messaging Server**: Per system requirements and specifications, there is a need to send push notifications to user devices. The specifics regarding

Cloud Messaging Server selection is platform dependent. For EASIER's architecture, any cloud messaging API that is accessible over HTTP requests is usable.

- **Weather Server**: Weather information is provided to user devices and may be factored in to Alert Rules and Alert generation. Any Weather API that is accessible over HTTP requests is usable.

- **Social Network Community**: A core feature of the EASIER system is the ability to distribute generated Alerts to opted-in community members. It is assumed that these community members are known, and that they are not expected to use any specific software for the project. This means some method of sending messages externally is needed such as SMS, Email, social media, phone calls, etc. For the EASIER architecture, as per the EASIER Web Server's design, this component must be accessible over HTTP request.

2.2.2 Component Interactions

This subsection describes the interaction between different core components of the EASIER System.

User Home

The user's home is furnished with power and networking capabilities. These are used to power internet connected Air Quality sensors.

User Device

The user's device sends a request to the EASIER Web Server repeatedly. The server responds with air quality information, generated alerts, and home data. The user's device also reports its own ID keys and usage information to the server. The user's device fetches weather information directly from the Weather Server. Push

notifications are sent to the user's device natively. The user's devices additionally may fetch air quality information in real time directly via the Air Quality Sensor API, or defer to the EASIER Web Server's last fetched air quality information datapoint.

EASIER Web Server

EASIER Web Server fetches air quality information from Air Quality Sensors and calculates Alerts using Alert Rules, using the database for Alert Rules, generating Alerts, and logging datapoints. EASIER Web Server fetches weather information from the Weather Server. EASIER Web Server triggers push notifications, and Social Network notifications as necessary. When an Alert is generated, it is provided to the User Device when a request is received.

Cloud Messaging Server

The Cloud Messaging Server receives HTTP requests from the EASIER Web Server and generates push notifications, as well as Social Network Community notifications. It is possible to separate these actions to different APIs if necessary.

Weather Server

The Weather Server is simply sent a request for up to date weather data. This weather data is displayed in the user-facing EASIER application on the user device.

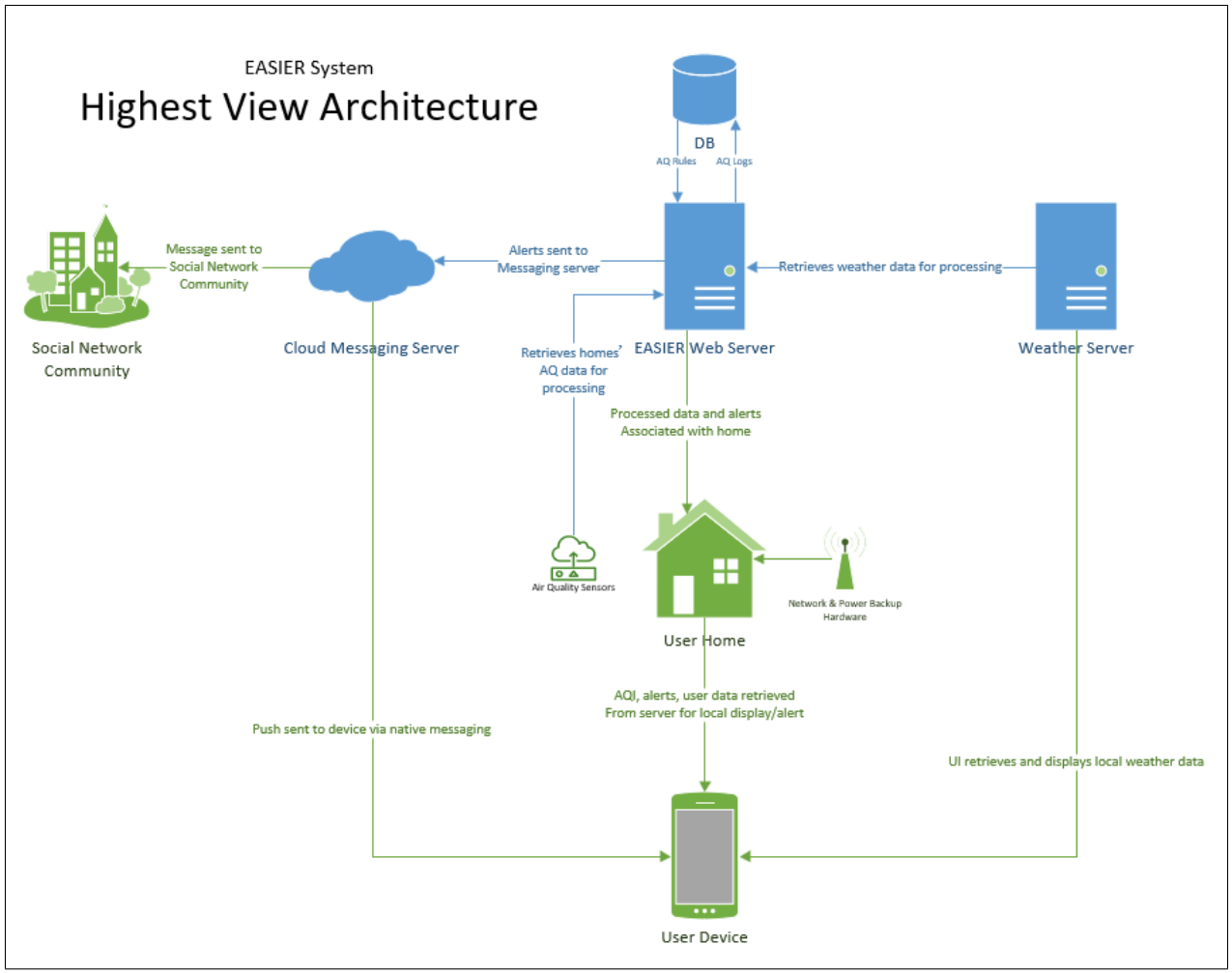

Figure 2.1: Highest View General Architecture graph

2.3 Server Architecture

The EASIER Server (or "EASIER Central Server") is the point to which all units "phone home", and the source of generated Alerts. The EASIER Server interfaces with multiple databases: to store sensor readings, to store heuristic data calculated based on these data, to create/update/delete Alerts, and to interact with Alert Rules. Further, the EASIER Server fetches sensor readings from the sensor units' API, and generates outbound push/SMS notifications. Additionally, since Alert Rules may need to be changed, and new units are brought online frequently, it is important that there is a way for (likely non-technical) team members to be able to access and update information in the system unsupervised. A pseudo-UML representation of the EASIER Server's general architecture can be seen in Figure 2.2 and is described below.

2.3.1 Architecture Description

Components

In the EASIER Server, there are six main components:

- **EASIER Web Server**: While the term *EASIER Server* and *EASIER Web Server* may be used interchangeably in this paper, in the context of this server-specific architectural description, "EASIER Web Server" refers to the server's component that makes content available on demand to users or devices over HTTP and the associated libraries and functions necessary to enable web-based communication. *EASIER Server* refers more broadly to all of the web-based, centralised components.

- **Alert Rules Database**: The Alert Rules Database stores rules for determining whether a given sensor reading generates an Alert or not. Alert Rules are given a sensor type (ie: CO2, Temperature, etc), an operator (current implementations

include *GREATER THAN*, *LESS THAN* and *EQUALS*) and a threshold value (set according to the sensor API's provided measurement units).

- **AQ Sensor Log Database**: The Air Quality Sensor Log Database stores raw air quality sensor readings for every unit's readings.
- **In-app Alert Interface**: The in-home devices actively query the EASIER Server for generated alerts and other necessary data - the In-app Alert Interface provides access to this data via HTTP requests.
- **Push Notification API Interface**: This interface generates external/native push notifications when required.
- **Social Network Alert API Interface**: This interface generates social network notifications when required.
- **EASIER Homes Panel**: The Homes Panel provides an access point for non-technical team members to onboard new devices, adjust Alert Rules, monitor generated alerts, test notification functionality, and generally engage with the EASIER Server.

2.3.2 Component Interactions

EASIER Web Server

The component described as *EASIER Web Server* describes the general framework that allows the scripts and databases to be accessed over the web. This component *is the server itself*, and as such interacts with all other components. In the current implementation of EASIER, the *EASIER Web Server* component refers to the Apache based web server accessed via shared web hosting. Through the server's automation, it uses HTTP requests to:

- Trigger push notifications via the Push Notification API Interface.

- Trigger social network notifications via the Social Network Alert API Interface.
- Access all databases (AQ Sensor Log Database and Alert Rules Database).
- Fetch sensor readings via the AQ Sensor Interface, and from the Weather Interface.
- Provide an interface for the EASIER Team Homes Panel to allow team members to update content.

Lastly, the EASIER Web Server handles all Alert generation, and calculates sensor heuristics.

Alert Rules Database

Generally, Alert Rules are defined manually. The Alert Rules Database thusly receives information from the *EASIER Team Homes Panel*. Additionally, functionality exists to track user acknowledgement of alerts by updating the database via received requests to the EASIER Web Server. The EASIER Web Server also retrieves Alert Rules Database values in the process of generating Alerts.

Air Quality Sensor Log Database

The AQ Sensor Log Database logs Air Quality sensor readings.

EASIER Homes Panel

The Homes Panel is used to allow team members to define and update rules, test other API connections (ie, verifying functionality of push notifications and in-app notifications), and to review data. The Homes Panel's dynamic frontend enables non-technical team members to engage with the system's databases easily. The current implementation uses Bootstrap for styling, powered by jQuery and AJAX to allow immediate updating of database values with dynamic responses. Other similar frameworks allowing for development of a responsive website capable of making HTTP

requests and dynamically updating the website with accessible styling would be able to be used instead of the Bootstrap/jQuery/AJAX framework in place now, which were selected due to their familiarity and compatibility. The Homes Panel acts simply as a frontend for the EASIER Web Server, with added system-testing functionality like automatic insertion into the Generated Alerts Database. The Homes Panel is designed to be used by all team members, including non-technical project members, making accessible, simple UI necessary. A more detailed description of the functions and implementation of the Homes Panel is found in Chapter 3.

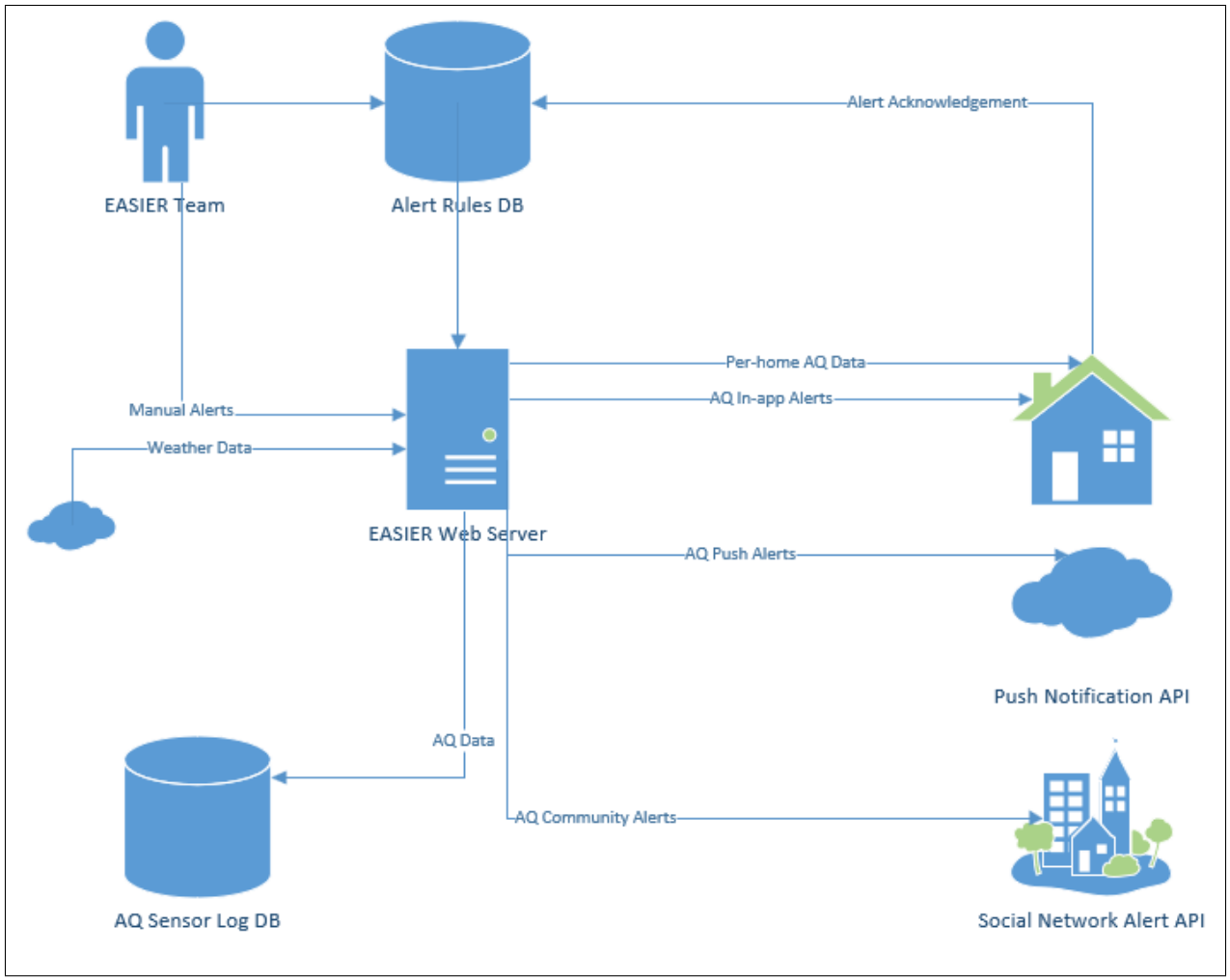

Figure 2.2: Server General Architecture graph

2.4 Device Architecture

The EASIER System relies on each home being provided a device that is capable of at minimum:

- The ability to download and run software from some centralised App Distribution platform.
- The ability to communicate over the internet with the EASIER Web Server.
- The ability to receive push notifications even when software is not active.

It is beneficial for the provided device to be easy to use for target demographics with as much functionality as possible for future expansion. It is for this reason that an Android tablet was determined to be optimal, at least for initial implementations, though similar devices would also be suitable like smartphones, iPads, convertible PCs, etc.

2.4.1 Architecture Description

Components

- **Provided Device**: Each home/unit is provided a device capable of the above functionality. For the initial EASIER implementation, this was a modern Samsung tablet running Android with Google services. The tablet is able to be pre-configured with EASIER's application and necessary API keys and accounts.
- **EASIER Web Server**: The EASIER Web Server is considered a component of the device's architecture as the tablet is regularly communicating bidirectionally with the EASIER Web Server. Without a connection to the EASIER Web Server, the software running on the device would not have full functionality.

- **App Distribution Platform**: Some form of app store/updating tool is typically included in consumer devices. EASIER relies on this updating capability to automatically push software updates containing critical changes like new features or bug fixes to devices.

- **Cloud Messaging Server**: Cloud Messaging services are typically an external library that must be implemented in software because they rely on system-level functions to enable background webhooks to generate push notifications. In the context of the EASIER project, each device generates its own Cloud Messaging ID, which is referenced by the Cloud Messaging API on the EASIER Web Server to generate push notifications on the device.

- **Weather Server**: Weather is provided as a convenience to the user via an external Weather Server.

- **Network and Power Backup Hardware**: Battery backup power supplies and network access points are provided to users to ensure full functionality even in extreme circumstances.

A pseudo-UML representation of the Device General Architecture can be seen in Figure 2.3

2.4.2 Component Interactions

Provided Device

The provided device interacts with all other components of the Device level architecture. In this context, the "Provided Device" refers to the EASIER software. The device reports usage, Cloud Messaging device IDs, and Alert acknowledgements back to the EASIER Web Server over HTTP. The device requests alerts and air data from the EASIER Web Server.

EASIER Web Server

The EASIER Web Server receives data reported by each device and stores it. When requested over HTTP by the device, the server provides air data and alerts as described in the Server General Architecture (Chapter 2.2).

Cloud Messaging Server

The Cloud Messaging Server triggers on-device push notifications when an alert is generated on the EASIER Web Server. For this to function, each device must report its Cloud Messaging device ID to the EASIER Web Server and enable necessary permissions.

Weather Server

The Weather Server returns realtime weather information to the device for the user's convenience when it is queried over HTTP.

App Distribution Platform

The App Distribution Platform ensures that the application regularly updates to the latest software version. This allows flexibility in implementation and updating of software.

2.5 Description of an Epoch

For devices to receive alerts, they must first be generated centrally. For alerts to be generated, data must be fetched from associated sensors and compared against defined alert rules. This procedure requires a combination of external API calls, database accesses, calculations, and read/write operations. In this project, we describe one cycle of all server-side processing as an "Epoch". In the current EASIER implementation, a Cron Job is used to run an Epoch every 5 minutes. In this section,

a high-level overview of the procedures in each step of an Epoch will be described. Later chapters will cover these operations' implementations in more detail. A pseudo-UML graph representation of an Epoch can be seen in Figure 2.4.

2.5.1 Steps in an Epoch

1. Find active devices.
2. Fetch all active devices' latest air data.
3. Determine system-wide heuristics.
4. Calculate device heuristics.
5. Fetch alert rules.
6. Process Alert Rules.
7. Generate and Process Alerts.
8. Store logs in database.

2.5.2 Step Details

Find Active Devices

Call the sensor API to get a list of all device IDs currently reporting measurements. Store these IDs temporarily and compare them to device IDs registered in the database.

Fetch All Active Devices' Latest Air Data

Call the sensor API to get a list of all sensors' most recently reported measurements. Map these measurements to associated devices/homes and store them temporarily. The previous API call step and this API step may be combined in practice.

Fetch System-wide Heuristics

Information regarding system-wide communication is stored in memory temporarily:

- Number of non-responsive connected devices.
- Number of responses.
- Total registered devices.
- "Did All Respond?"

Calculate Device Heuristics

Sensor data from the current epoch is compared against the previous epoch to determine for each sensor if the given value increased, decreased, or was the same. These results are stored in memory temporarily. Once per-device heuristics are calculated, more system-wide heuristics are calculated to determine the system-wide count of number of each type of sensor whose measurements increased/decreased.

Fetch Alert Rules

Alert Rule objects (a data structure defined in a later chapter) are generated after retrieving Alert Rule thresholds from a database. See the Alert Rule Object diagram.

Process Alert Rules

Alert Rule objects' compare function(s) are run against each device's sensor values. See the Alert Rule Holder Object diagram.

Generate and Process Alerts

Alert Rule objects' compare functions generate Alerts when the threshold is triggered. Generated Alerts are stored in the database of active alerts. If applicable, at this stage, any external notification actions are triggered via their respective APIs (push notifications, SMS notifications)

Store Logs in Database

System-wide and per-device heuristics, as well as all sensor readings, are stored in a database for later analysis and comparison.

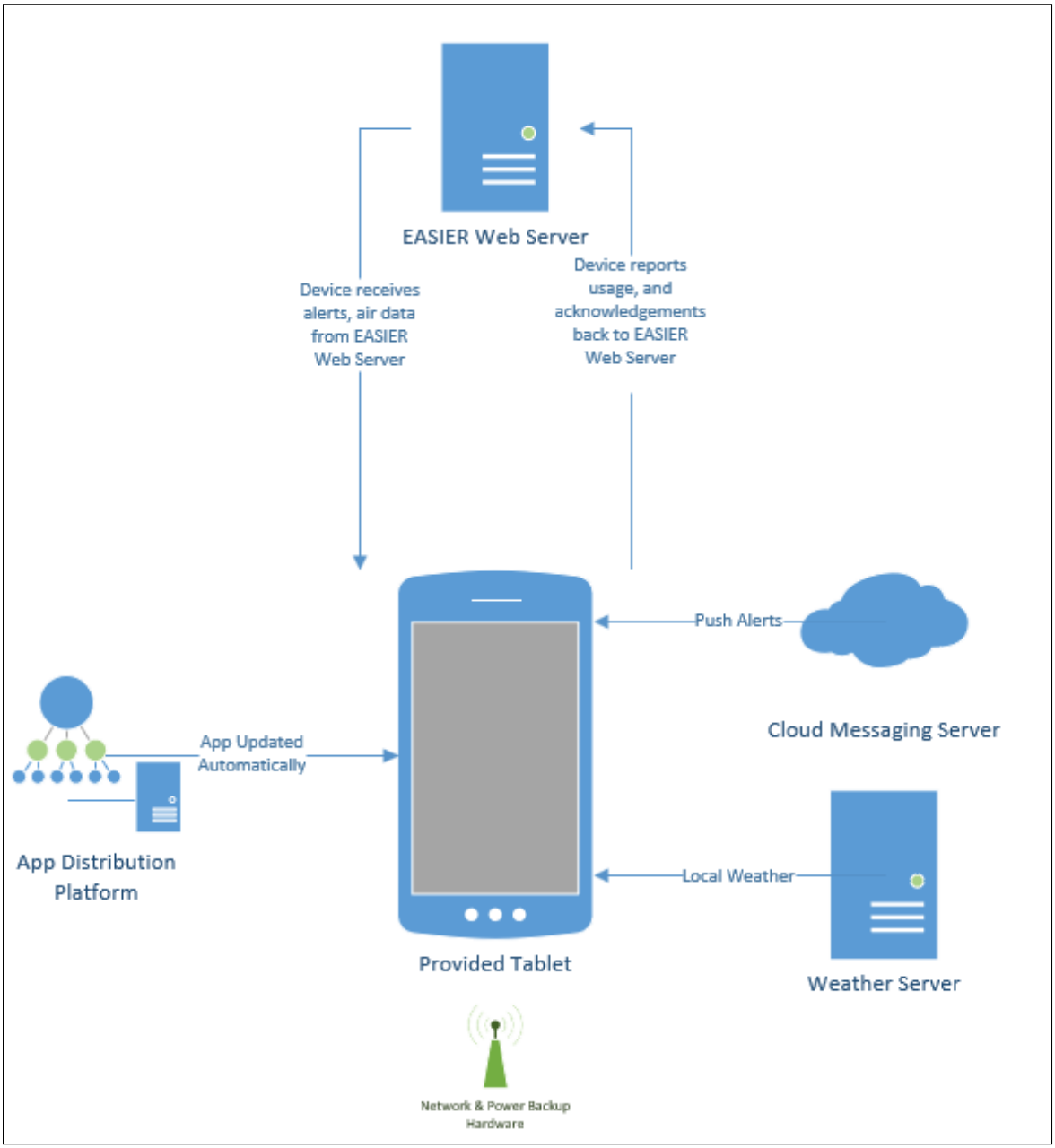

Figure 2.3: Device General Architecture graph

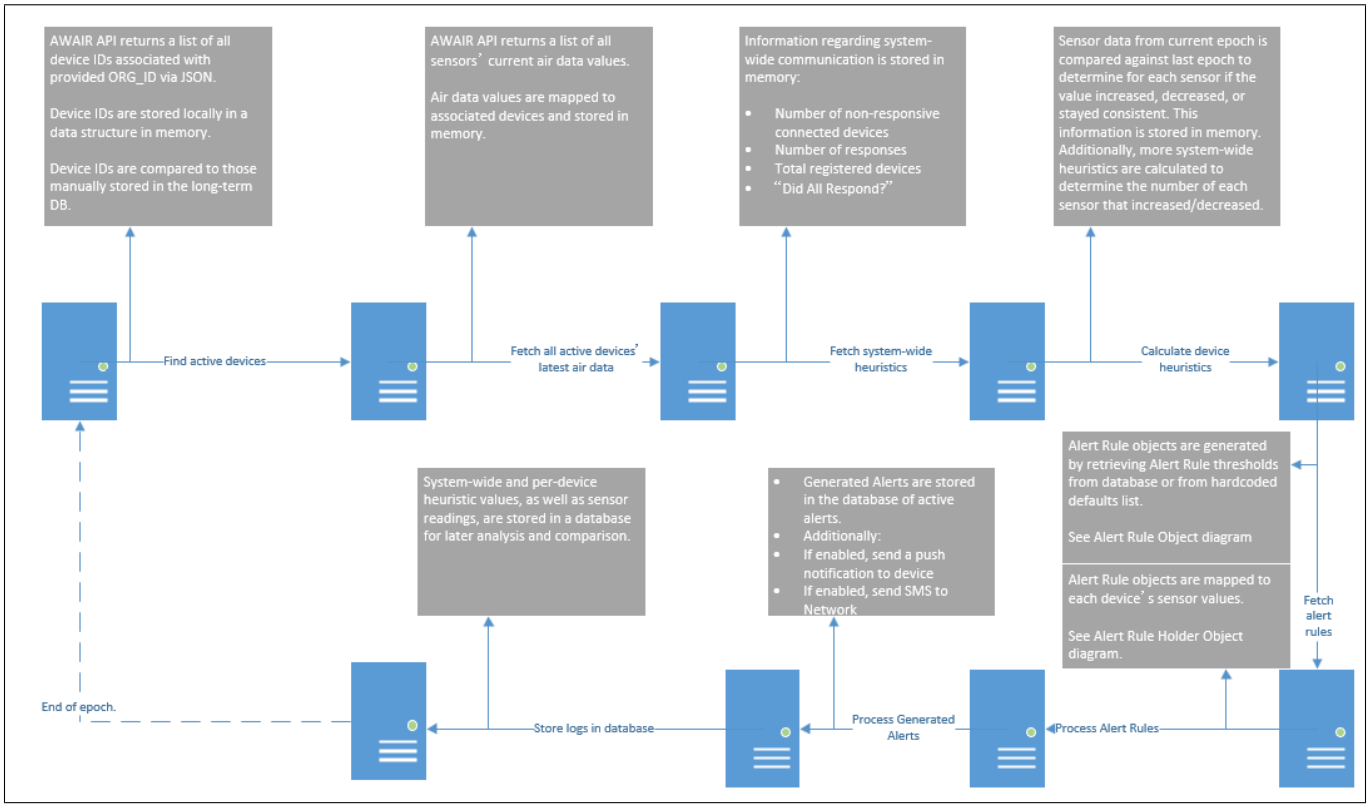

Figure 2.4: Epoch General Architecture graph

Chapter 3

System Implementation

3.1 Implementation Details

3.1.1 Components

User Home

- **AWAIR Air Quality Sensors** provides air quality information via their public API. By using HTTP requests, a home's current temperature, humidity, CO2 level, TVOC level, and PM2.5 level can be recorded.
- **Battery** backup UPS provides back up power for all other components.
- **Verizon Hotspot** acts as a wireless access point for all other components.

Device

Android tablets produced by Samsung are provided to each home. Each device is linked to a Google account created by the development team. Initially, accounts had to be managed by those involved in the project so automatic updates could be provided to users via Google Play's internal alpha/beta testing distribution framework, which requires manual entry to invite a new approved Google account. These Google accounts had to be manually opted in before the app could be

downloaded and enrolled in the testing distribution channel. Due to this complexity, it was decided to instead simply publish the application publicly to the Google Play store, which would be manually downloaded on each device. Though application updates now must go through Google Play's update procedure meaning updates can take a day or so to propagate (as opposed to nearly instantaneously becoming available on the testing channel) this was deemed preferable to the high overhead of managing and enrolling devices on a testing channel. Still, each device has an associated, project created Google account. This allows features like security policies and remote device management through Android's features, especially via Google Workspace. Samsung's tablets were selected due to their high quality and relative affordability, as well as their offering of easy development opportunities via their distribution of Android. However, other tablets running Android like Google's Pixel Tablets, or even Apple's iPads or Microsoft's Surface Tablets would be equally easy to implement. Further, with the EASIER application being implemented in Flutter, little work would be necessary to swap to another device platform.

EASIER Web Server

The EASIER Web Server handles all database interactions, fetches air quality sensor values, calculates alerts, and communicates with all external APIs used for this project. The device requests necessary data from the EASIER Web Server which it displays to the user. The implementation and functionality of the EASIER Web Server will be described in a later section - this section serves to briefly describe the details of the server's software.

- **Apache** shared hosting via Hosting24 is used for the EASIER Web Server. There are currently no high processor loads or memory intensive operations running on the server, so the cheapest option is being used. Outside of a few gigabytes of storage space for collected data and interface elements, nearly any modern web hosting platform would be sufficient for the EASIER Web

Server in its current implementation, assuming it supports the other listed software (which are all commonly used). Other common frameworks like Nginx, Microsoft's IIS, for even Node.js would be more than capable of fulfilling this role, but Apache was selected due to its availability and familiarity.

- **PHP** is the programming language used for the majority of the EASIER Web Server. It is used to interface with the databases, to communicate with external APIs, to send/receive information to devices, to log data, and nearly every task the EASIER Web Server performs outside of front-end web content that requires JavaScript-based realtime updates. The EASIER Web Server does not use any "unusual" features of PHP, and should be compatible with any modern version of PHP. PHP was selected due to familiarity and compatibility with Apache web servers.

- **Cron** scheduled tasks are used to run a system epoch (described in Fig 2.4) every five minutes. Cron is a UNIX tool that executes a PHP file with a scheduled frequency. Other task scheduling tools would be able to adequately perform this task. Cron was selected due to its inclusion with Hosting24's shared web hosting plans.

- **MySQL** databases store sensor readings, alerts, logs, home info, etc - everything recorded by the EASIER Web Server. The databases' structure is described in detail in the database section 3.2. Outside of a few relational SELECT statements, no advanced features of MySQL were used, meaning MySQL could be easily substituted for other database platforms with simply changing a few lines in library code used for retrieving/storing data. MySQL was selected due to its availability via Hosting24, as well as familiarity. PHPMyAdmin (a PHP based MySQL database management tool) is provided via Hosting24 and was used for initial setup, modification, and exporting of MySQL databases.

- **Bootstrap** provides the UI framework for the EASIER Web Server's "Homes Panel", the user facing website allowing entry and modification of database data by non-technical project members. Bootstrap provides a quick and easy way to implement responsive websites using CSS. It was selected for its familiarity, simplicity, and abundance of resources (Subrahmanion et al. [2021]).

Cloud Messaging Server

One of the most important features of EASIER is the ability to send push notifications to users' devices when an Alert Rule is violated and an Alert is generated. Since the EASIER application is currently only deployed on Android, the obvious choice for implementing push notifications is Firebase Cloud Messaging (formerly Google Cloud Messaging). FCM can also be used for non-Android platforms. A central FCM account was created to provide API tokens for internal operations, and each device reports its own FCM device token that is generated at runtime and sent to the EASIER Web Server when the application is first run. When an Alert is generated on the server, the associated FCM device token is fetched and a push notification is sent to the device by querying FCM's HTTP-based API. While FCM is the optimal choice for an Android application, other notification services like OneSignal or Amazon SNS might be preferable should the project's scope expand to include other categories of devices.

SMS Messaging Server

Another important feature of EASIER is the ability to send text messages to enrolled Social Network Community contacts when an Alert is generated. Twilio's HTTP-based API is called during the Alert Generation stage of an epoch. If the home receiving an alert has an associated phone number, Twilio will send an SMS message to the number. Some difficulties were encountered when initially integrating Twilio functionality, as recent US regulation has required a fairly extensive verification

process to acquire and use toll-free phone numbers as well as to send text messages via a web-based service. Twilio was selected because of its sufficiently inexpensive options for pay-as-you-go SMS messaging, making it ideal for the EASIER project's needs. Other options like Plivo capable of sending SMS messages via a web-accessible API would be equally usable.

Weather Server

EASIER provides users a summary of local weather conditions in the application interface. As of this writing, there are near-term plans to implement realtime weather based alerts (for emergencies like floods, storms, poor air quality, pollen, etc) meaning the service used may change due to future requirements. Currently, only basic information like temperature and humidity is displayed in the application, so any compatible API would be sufficient. Open-Meteo was selected as the weather provider due to its free, simple API that reports JSON data via HTTP request. Other platforms like AccuWeather, WeatherAPI, OpenWeatherMap, etc - would be equally suitable.

Social Network Community

Recruitment for homes involved in the EASIER project has been initially limited to Tennessee, as local volunteers and project members are able to physically train users, transport devices, and provide other services necessary for the EASIER project. Recruitment has been mostly handled by a local non-profit organisation via word of mouth. Once a home has been recruited, surveys are given to the home to provide information necessary for the project's operation. One of these surveys includes establishing a potential "Social Network Community" - which is defined as one or more individuals willing to receive text message notifications generated by EASIER's alerts, and hopefully provide feedback on the experience later on. There are plans to expand the Social Network Community to include local authorities like paramedics

and government officials, as well as provide other means of contact and involvement for those opted-in.

Homes Panel

The Homes Panel provides a web frontend for team members (especially non-technical team members) to interact with the databases, data, and APIs used by the EASIER Project. Bootstrap based frontend is generated via server-side PHP code. The site dynamically updates page content using AJAX and jQuery libraries for JavaScript, allowing more intuitive realtime interactions for non-technical users. These frameworks were selected because of their compatibility with the EASIER Web Server, as well as general familiarity. A detailed description of the Homes Panel can be found in the EASIER Homes Panel section of this chapter (Section 3.4)

3.1.2 Component Interactions

This section describes how the actual implementations of these components interact with each other in the EASIER system, mirroring the architectural description of these interactions in Chapter 2.

User Home

Each home is equipped with:

- An uninterruptible power supply (a.k.a UPS) with a battery backup, and a 4G enabled router, ensuring all devices remain connected with power.
- An AWAIR Element sensor array, that is connected to the 4G enabled router and UPS battery power supply.
- An initialised Android tablet, ready to use the EASIER application.

Information to identify all devices remotely is stored in the EASIER Web Server for later retrieval.

User Device

- Each device is setup using a Google Account created solely for its associated home.
- The EASIER application records and reports the device's associated AWAIR IDs and FCM tokens, allowing all interactions to be logged and associated with the home.
- The device is (at least initially) connected to the provided 4G enabled router allowing constant connectivity.
- Each device retrieves alerts from the EASIER Web Server independently. Realtime AWAIR data is retrieved from the AWAIR API via the EASIER application, rather than over a local network or via AWAIR's own applications.

EASIER Web Server

- The EASIER web server receives requests for generated alerts from devices periodically.
- Alerts are generated every few minutes using a scheduled Cron job.
- The EASIER Web Server has libraries for database interaction, allowing the Homes Panel to fetch and display information easily.
- API calls are generated to Twilio and Firebase for external notifications via the EASIER Web Server.
- The EASIER Web Server uses shared webhosting, meaning development uses a combination of a host-provided cPanel environment and a remote FTP-enabled environment.

Homes Panel

- The Homes Panel can fetch and update/insert data using the EASIER Web Server's database libraries.
- Functionality also exists to manually trigger API calls to test push and SMS notifications.

Cloud Messaging Server

- All cloud messaging is generated by calls from the EASIER Web Server to Twilio or Firebase's APIs.

Weather Server

- Data is fetched from Open-Meteo via their API by the EASIER application periodically.

3.2 Server Implementation Details

3.2.1 Overview of an Epoch

The term "Epoch" is used to describe all actions that occur periodically on the EASIER Web Server. An Epoch is triggered by a Cron job, but is fundamentally one large PHP function that calls many other PHP functions that perform all tasks that the server is expected to do. This includes fetching air quality data from AWAIR, processing the data against the Alert Rules defined in the database, sending push notifications; essentially everything involved in the retrieval, processing, and distribution of data on the server. Chapter 4 offers a detailed breakdown of every step of an Epoch; but a general understanding of the term as used in this paper is required to understand this section.

Objects

The EASIER Web Server uses a number of objects (generally, PHP classes) to simplify interactions between the functions that process data. In general, all objects serve as an intermediary step to the end goal of creating an Alert that can be inserted in the database. Most of these objects are helper prerequisites for the Alert Rule Object as shown in Figure 3.1. The Alert Rule object is passed an Alert Sensor Type Object, an Alert Condition object, and a threshold value. This object later is bound to a variable that receives a given device's relevant AQI sensor value as fetched via AWAIR. Each Epoch handles the generation of these objects and their bindings, and later calls functions for checking whether the values stored in the object result in an Alert being generated. When they do, a separate Alert object storing the same information is generated, to be inserted into the database later in the Epoch.

Libraries

The backend code used by an Epoch of the EASIER Web Server is broken down into multiple categories (called "libraries"), most of which serve as helpers in accomplishing one specific one individual task. These libraries are separated into the following categories: AWAIR libraries, Cron libraries, Database libraries, FCM libraries, Summary, and Alert libraries. Since there are a few dozen "libraries" being used every Epoch, a full description of every library is beyond the scope of this paper, but a few are described below:

- **Alert Library** handles enumeration of sensor names, condition operations (less than, greater than, etc), and most importantly defines the AlertRule class as well as the functions required to map an AlertRule to a sensor value.
- **AWAIR Library** handles all communication with the AWAIR API. This library includes functions that fetch a list of all devices' identification information, retrieve the latest sensor readings, and calculate all device data heuristics (see: Section 3.3.2). Functions that generate all default Alert Rules

(those not specified in the Alert Rules Database), as well as the functions that generate Alert Rules from the database, map values to the Alert Rules, call the comparison functions, generate Alert objects, and store Alert objects via the Database Libraries are also found here.

- **Database Libraries** provide helper functions to communicate with all of EASIER's databases. There are also functions to do specific operations on the data in the database. Most importantly, these libraries define the functions that are used as endpoints for API calls to the EASIER Web Server. When the application sends a request to the EASIER Web Server for retrieval or updating of data, it likely goes directly to one of these libraries, which handle HTTP based communication, define the way the data is processed and transmitted, calling of other necessary helper functions, and error/success reporting.

- **FCM Library** handles communication with Firebase Cloud Messaging over HTTP.

- **Summary Library** is used by the Homes Panel to process large amounts of heuristics in the database. Periodic air quality summaries will be provided to each home, which require the segmentation and processing of thousands of database entries. Rather than doing this summary generation manually, helper functions to do core tasks like finding a given home's daily/weekly/monthly highs, lows, averages, medians, etc - was implemented and is used to generate human readable data to be entered into a physical document to be mailed to each home.

3.2.2 APIs

The main APIs accessed by the EASIER Web Server are AWAIR, Open-Meteo, Twilio, and Firebase. Luckily, all of these APIs offer simple HTTP request based

APIs, and generally report data in a simple JSON format. This means all API integration on the server side is handled via simple PHP cURL functionality.

3.3 Databases

3.3.1 Overview

This section aims to serve as a detailed description of the structure and usage of the EASIER Database in its server-side implementation. Figure 3.2 shows the relationships between tables in the database, as well as the elements of each table and their associated data types. Each table will be described separately in more detail below.

The EASIER database uses MySQL, allowing for easy interfacing with the EASIER Central Server's PHP scripts. MySQL is sufficient for the scale of the EASIER project in its current scope. Other database systems offer different combinations of benefits and drawbacks in regards to performance, but are irrelevant for a project of this size. Most functionality of the EASIER database could be easily transferred to a different database framework with only minimal changes, predominately in regards to syntax of inserting and selecting data from the database.

Figure 3.1: EASIER Web Server's Alert Rule object.

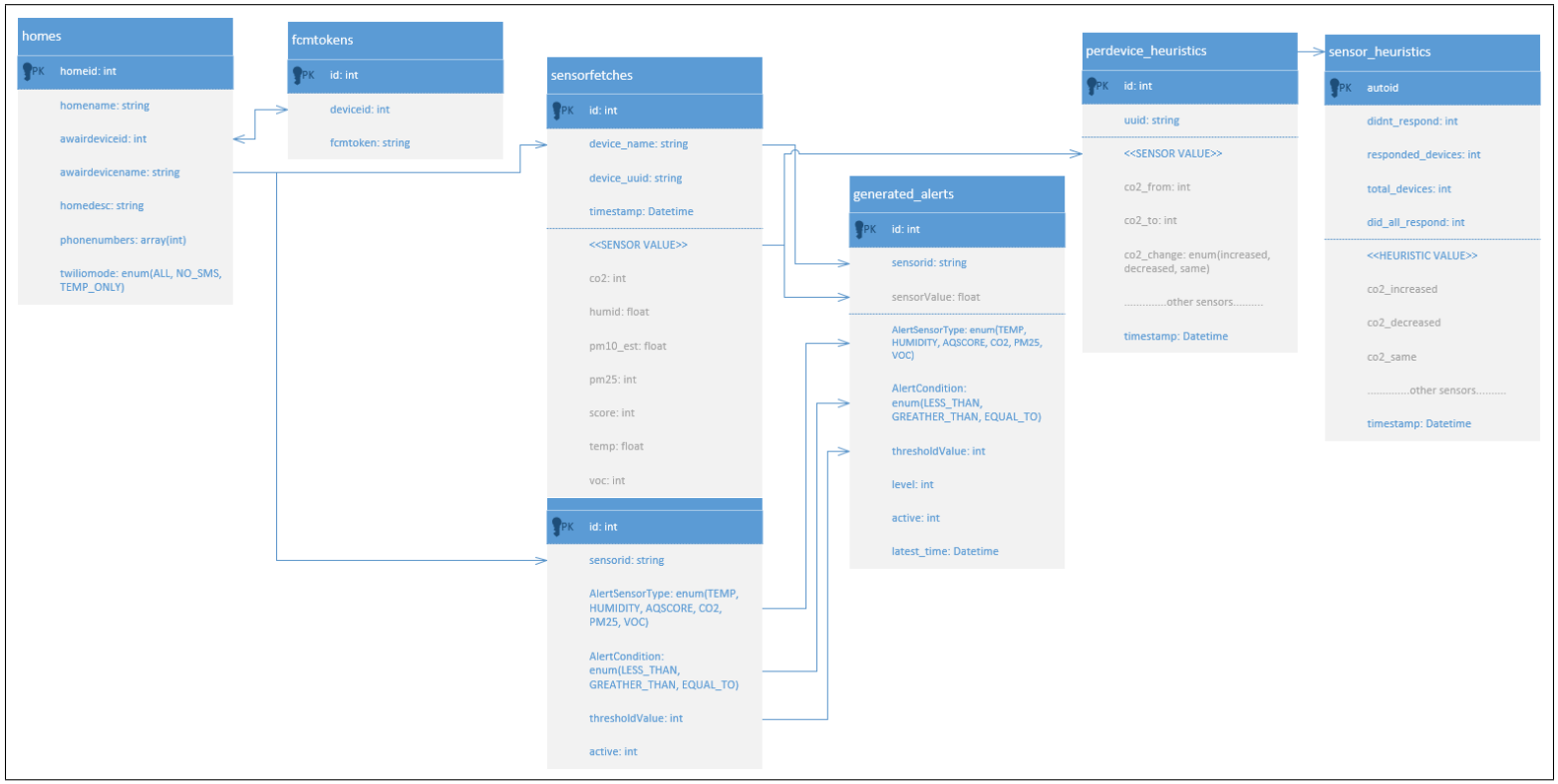

Figure 3.2: Database Structure and Relationships

3.3.2 Internal Tables

This subsection describes tables that are populated entirely internally. Typically no human user will be adding entries directly to tables considered to be "Internal". However, these may rely on data from other tables that are not populated entirely internally.

System Logs Table

The Logs table is only used to record when a server-side epoch occurs. An entry is inserted every epoch storing a Datetime timestamp. This table was not included in Figure 3.2 due to its simplicity.

Sensor Fetches Table

The Sensor Fetches table is arguably the most important component of the entirety of the EASIER system. An entry is created in this table for every measurement for every device. As of writing, it contains over 1.5 million entries. One entry in this table contains a device name and all associated sensor readings, as well as a timestamp. In the current implementation of EASIER, the device name is identical to the name automatically generated by the AWAIR Sensor API, which is the same as the device's UUID.

Table Elements

- *id* - an integer serving as the Primary Key (PK) for this entry, auto incrementing.
- *device_name* - a string storing the ID of the device reporting this entry's data. In this implementation, the format is always "awair-element_n" where n is a uniquely identifying number automatically generated by AWAIR on device onboarding, typically 5 or 6 digits.

- *device_uuid* - a string storing the UUID of the device reporting this entry's data. In this implementation, the device_name is the UUID, so these columns are the same.

- *timestamp* - a Datetime timestamp automatically generated on MySQL insertion to record the exact time this entry was inserted in the database. Formatted as a YYYY-MM-DD HH:MM:SS timestamp.

- *co2* - an integer to store the device's CO2 concentration measurement. AWAIR reports CO2 concentration in parts per million (ppm). Expected values fall between 500ppm and 4500ppm.

- *humid* - a float to store the device's humidity reading. AWAIR reports relative humidity percents as floats to hundredths precision. Expected values between 40.00 and 80.00.

- *pm10_est* - a float to store the device's PM10 reading. AWAIR reports estimated particulate matter less than 10 microns in diameter via its PM2.5 sensor and reports them to four decimal places.

- *pm25* - an integer to store the device's PM2.5 "fine dust" reading. AWAIR reports PM2.5 concentrations in micrograms per cubic metre. Expected values between 0 and 150.

- *temp* - a float to store the device's temperature reading. AWAIR reports temperature to hundredths of a degree Celsius.

- *voc* - an integer to store the device's Total Volatile Organic Compound concentration reading. AWAIR reports TVOC concentration parts per billion (ppb). Expected values between 0 and 25000.

Per-device Heuristics Table

The Per-device Heuristics table stores summary heuristics calculated during an epoch, comparing the current sensor reported data (ie, that stored in the Sensor Fetches table) with the sensor reported data from the previous epoch. In general, the goal of this table is to record whether a device's reported sensor values increased, decreased, or stayed the same - as well as the previous and current sensor values.

Table Elements

- *id* - an integer serving as the PK for this entry, auto incrementing.
- *uuid* - a string recording the device's device_name value, equivalent to Sensor Fetches "device_name".
- *co2_from* - last epoch's CO2 value, the same as in Sensor Fetches.
- *co2_to* - this epoch's CO2 value, the same as in Sensor Fetches.
- *co2_change* - an enumerated string calculated by comparing co2_from and co2_to. Possible values are: "increased", "decreased", "same".
- *[sensor_name]_[from,to,change]* - following the same pattern as above, these columns represent the previous sensor value and current sensor value as stored in the Sensor Fetches table, as well as its associated enumerated string to represent its comparison. The possible sensor_name values are the same as those in the Sensor Fetches table.
- *timestamp* - a Datetime timestamp automatically generated on MySQL insertion to record the exact time this entry was inserted in the database. Formatted as a YYYY-MM-DD HH:MM:SS timestamp.

Sensor Heuristics Table

The Sensor Heuristics table stores per-epoch calculations that process Per-device Heuristics values. An entry serves to show for this epoch how many devices had a given sensor's value increase/decrease/stay the same. Using these values, other metrics like device responsiveness are calculated.

Table Elements

- *didnt_respond* - an integer storing shows the number of registered devices (found from the Homes table) that did not report data for the current epoch.
- *responded_devices* - this integer shows the number of devices that did report data for the current epoch.
- *total_devices* - an integer storing the number of registered devices found from the Homes table. total_devices = didnt_respond + responded_devices.
- *did_all_respond* - an integer treated as a bool that is 1 if responded_devices equals total_devices and 0 otherwise.
- *co2_increased* - an integer storing the number of devices this epoch that reported a CO2 change of "increased".
- *co2_decreased* - an integer storing the number of devices this epoch that reported a CO2 change of "decreased".
- *co2_same* - an integer storing the number of devices this epoch that reported a CO2 change of "same".
- *[sensor_name]_[increased,decreased,same]* - following the same pattern as above, these columns represent the number of devices this epoch that reported a given change. The possible sensor_name values are the same as those in the Sensor Fetches table

- *timestamp* - a Datetime timestamp automatically generated on MySQL insertion to record the exact time this entry was inserted in the database. Formatted as a YYYY-MM-DD HH:MM:SS timestamp.

Generated Alerts Table

This table contains an entry for every currently active generated alert. When a device has a sensor value calculated to be a violation of an Alert Rule, an Alert is generated in-memory during the epoch's Alert Generation phase. An entry is created to store the sensor type, the threshold operator, the threshold, and the value that generated a violation of an Alert Rule. If the device already has an Alert entry for this combination of sensor type, condition, and threshold, the existing entry is set to active if inactive, or its severity is increased and the threshold and sensor values are updated if it is active.

Table Elements

- *sensorid* - a string storing the id of the device reporting the data that generated this alert. This is the same as the Sensor Fetches device_name.

- *AlertSensorType* - an enumerated string storing the type of the sensor whose measured value led to the generated Alert. The possible values are the same as those in the Sensor Fetches table, though the enumeration is represented slightly differently as they are generated via the Alert Rule framework.

- *AlertCondition* - the enumerated string representing operator used for comparison in the Alert Rule that led to Alert generation. Possible values are "LESS_THAN", "GREATER_THAN", "EQUALS".

- *thresholdValue* - the integer value of the threshold used for comparison in the Alert Rule that led to Alert generation. This value is assumed to be in the same unit as the readings in Sensor Fetches' associated sensor type.

- *sensorValue* - the float value of the sensor reading that led to Alert generation.
- *level* - an integer that is incremented every time an Alert of a same type that already actively exists in the table is generated.
- *active* - an integer treated as a boolean to represent whether the Alert is active (1) or inactive (0). An Alert typically is made inactive when it is acknowledged by a user.
- *latest_time* - a Datetime timestamp automatically generated on MySQL insertion to record the exact time this entry was inserted in the database. Formatted as a YYYY-MM-DD HH:MM:SS timestamp.

3.3.3 External Tables

This subsection describes tables that may be populated externally. More generally, these tables receive input either directly from a user via manual insertion via a database management system, via the EASIER Homes Panel website, or from specifically triggered user interaction with the EASIER Application.

Homes Table

This table stores home-identifying information. This information is collected manually when a new device (and thus, home) is on-boarded during the device install and AWAIR sensor onboarding processes. As soon as the data is input in the Homes table, the device becomes available to the system as a whole, meaning it can participate in Epochs' activities like Alert Generation. The information is typically initially entered via the EASIER Homes Panel website. Some information in an entry remains unchanged, while some fields are updated later on.

Table Elements

- *homename* - a string that can be thought of as a "nickname" for the home in question used mostly when a human user wishes to update some of its other associated data. Most devices follow the naming scheme of "Element xx" where "Element" comes from the AWAIR Element sensor name and xx is the ordinal number of the home's device bundle's setup. The one important exception is the lab device, named "Perkins Hall" - used for testing, requiring special treatment as it is not an AWAIR Element device (requiring manual overrides in many parts of the code).

- *awairdeviceid* - an integer that stores the numeric portion of the AWAIR sensor's device_name like in Sensor Fetches. For a sensor with a device_name of "awair-element_123456", this integer would be "123456".

- *awairdevicename* - a string that stores the AWAIR sensor's device name, and is the same as the device_name in Sensor Fetches.

- *homedesc* - a string string that stores a human readable description of the home and can be used as needed for team members to add notes or otherwise. Generally, this string contains "Home [awairdeviceid]" by default.

- *homeid* - an auto incrementing integer that serves as this table's PK and is used by scripts to associate a given home with a device bundle. As it is possible that a home's device bundle would change over time, so data can be persistently associated with a home, this additional identifying number is used in some cases.

- *phonenumbers* - a string string that is used to allow human users to enter phone numbers of opted-in social network contacts who will receive SMS notifications. These phone numbers are added manually in a single comma separated string.

- *twiliomode* - an enumerated string which indicates whether a given home should attempt to send SMS messages to the opted-in social network contacts

when an Alert is generated. Possible values: NO_SMS, ALL, TEMP_ONLY. TEMP_ONLY is used for homes that should receive only urgent low temperature alerts.

Alert Rules Table

This table allows per-home overriding of Alert Rule thresholds. Since Alert Rule thresholds are typically hard-coded, this table was added to enable the addition of a feature added to the EASIER Homes Panel website enabling human users to add per-home, per-sensor overrides (and additional Alert Rule thresholds) as necessary, often required due to the existence of homes with particularly unusual or inconsistent sensor measurements.

Table Elements

- *id* - an auto-incrementing integer that functions as this table's PK.
- *sensorid* - a string storing AWAIR sensor device name being targeted for this override. This is the same as the home's awairdevicename in the Homes Table and the device_name in the Sensor Fetches table.
- *AlertSensorType* - an enumerated string representing the sensor type for this override. Its format is the same as the AlertSensorType parameter in the Generated Alerts table.
- *AlertCondition* - an enumerated string that represents the comparison operator for this override. Its format is the same as the AlertCondition in the Generated Alerts table.
- *thresholdValue* - an integer representing the comparison threshold for this override. Its format is the same as the thresholdValue in the Generated Alerts table.

- *active* - an integer that acts as a boolean to indicate whether this override should be processed (1) or ignored (0).

FCM Tokens Table

This table stores FCM Tokens that are reported to the server when the application is first launched after being manually paired to an AWAIR device. Google Firebase Cloud Messaging requires a token that is generated at runtime in an Android application to be provided to their API when a push notification is to be sent to said device. This token is stored locally on the Android application, and then sent to the EASIER Web Server, along with the locally stored paired AWAIR device id.

Table Elements

- *id* - an auto-incrementing integer that functions as this table's PK.
- *fcmtoken* - a string that stores the FCM token as reported to the server by the Android device.
- *deviceid* - an integer that is the numeric portion of the AWAIR sensor's device_name like in Sensor Fetches, the same as the awairdeviceid in the Homes Table.

3.4 EASIER Homes Panel

This section will describe the requirements, functionality, and implementation of the EASIER Homes Panel. The EASIER Homes Panel serves to provide a browser-based system for interacting with the databases, APIs, devices, and data involved in the EASIER project. The Homes Panel's creation came out of a need that arose only after the project had already begun. Due to limitations in home participant recruitment and availability for experienced team members to set up devices, the onboarding process was gradual, meaning across many months the total 50 homes

were onboarded sometimes weeks apart. The process to onboard a device requires manual logging of device IDs, serial numbers, setup of accounts, and entry into the EASIER Databases. Initially, the database entry tasks could only be performed by directly accessing the database control panels - by logging in to the shared hosting provider, accessing PHPMyAdmin, navigating to the database and relevant table, and manually inserting the data one at a time. This process required a technical understanding of the implementation of the system and the way the database was structured, as well as a knowledge of how to interact with MySQL database panels. Additionally, direct access to the shared hosting is required to access the MySQL database panels, meaning anyone with access to the databases would have full access to all source code, and potentially irreparably dangerous actions on the account as a whole. The EASIER Homes Panel began as a way to allow non-technical project team members to complete the onboarding process entirely independently. Later, it evolved to offer a number of other features, and continues to serve as the main method that non-technical project team members engage with the technical aspects of the project.

3.4.1 Requirements

- Accessible via a normal web browser
- Intuitive design so training is not required
- Easy to use, requiring few steps to accomplish common tasks
- Allow easy access to necessary actions while restricting access to dangerous actions

3.4.2 Technical Details

Bootstrap was used as a free and easy to use UI framework for designing the website, which is hosted on the same shared web hosting as the EASIER Central Server. Many

elements on pages are dynamically updated with data (or dynamically submit data) to the server, accomplished via jQuery and AJAX based HTTP requests, handled by custom JavaScript code. Various freely available libraries were used for features like advanced charting and displaying Lottie animations on the pages.

3.4.3 Functionality

Add and Update Home Information

The primary functionality of the EASIER Homes Panel is to allow the easy addition and modification of homes' information into the database (thus the name "Homes Panel"). This functionality exists on the main page of the Homes Panel, as shown in Figure 3.3. The large table in the centre of the page shows a (dynamically sorted and updating) list of the already onboarded homes. The green "add new home" button adds a new, empty home. Any entry in the table can be edited directly as if it were a spreadsheet, and changes are automatically saved to the database via HTTP request, also providing the user a visual cue to indicate success (or failure should it occur). The editable columns include:

- Home Name - an internal name used by the team to refer to a specific home.
- Phone Numbers - Social network phone number contacts for SMS via Twilio.
- AWAIR ID - the ID number entered in the user's settings on the Android device.
- AWAIR Name - the full UUID of the device associated with the user.
- Home Description - used as a note to freely store any relevant information.
- Home ID - an auto incrementing, unique ID.
- FCM Token - auto generated Firebase token as reported by the device.
- SMS Enabled - indicates whether this home has SMS messaging features enabled..

There is also a "Home Actions" column, which offers three clickable buttons that trigger server-side actions.

Manage Alerts Button

The "Manage Alerts" button navigates to a separate page. The page contains a similar table, except offering a list of all currently active Alerts for the specified home. This page allows the deletion of an Alert, as well as the manual insertion of an Alert with custom specified sensor/condition/threshold/measured values, to simulate an Alert for a home even when one is not generated otherwise. This feature was implemented during early in-home testing, where a team member would be present in the home with a participant. A manual alert would be triggered, allowing the team members to observe and receive feedback regarding the Alert process.

Trigger SMS Button

There was a need to test SMS functionality before SMS functionality was integrated as a substep of the Alert process. This button, when clicked, triggers a Twilio SMS message to the home's specified phone number(s) with a test message via a dynamic HTTP request.

Trigger Push Button

Similar to the Trigger SMS button, this button triggers a test push notification via Firebase, accomplished with a dynamic HTTP request.

Threshold Manager Button

This button navigates to a separate page, which allows the addition or removal of "Alert Rule Threshold Overrides" on a per-home basis. Every home in the system has a default set of Alert Rules, designed in accordance with the team's specifications. However, it was found that some homes generate far more alerts than others, or

that at certain times of year there was a need to change thresholds. In general, due to a combination of factors (ranging from lack of established literature, to lack of consensus), the decision was made to implement a feature that allows team members other than the software developers to actively determine the implemented Alert Rule thresholds.

Summaries Button

The Summaries button navigates to a page that allows the generation of human-readable summaries of a home's sensor readings over a given time range, and realtime creation of associated graphs. This functionality will be described more later.

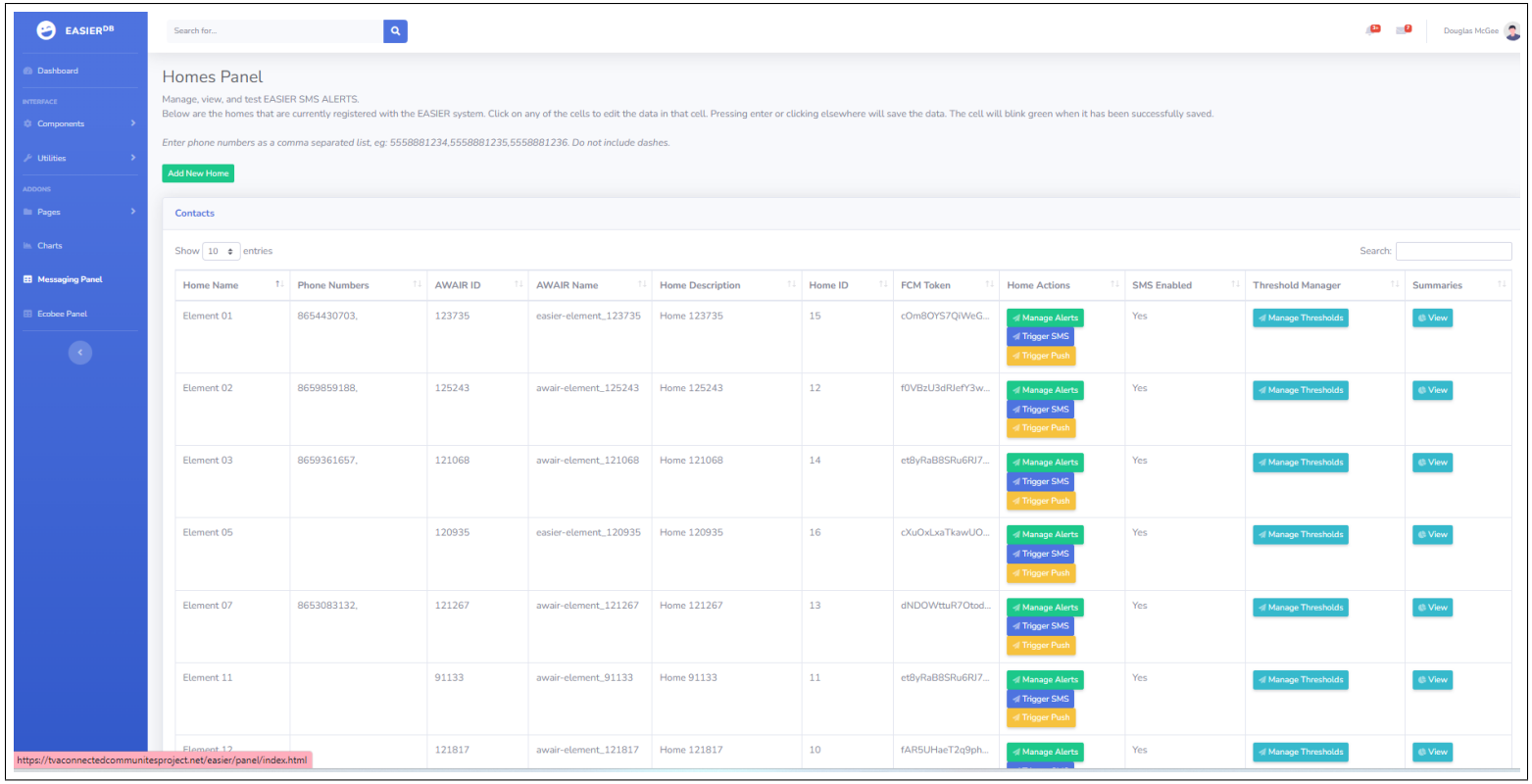

Figure 3.3: EASIER Homes Panel - main page

Chapter 4

An Epoch in Detail

"A Day In The Life of EASIER"

As mentioned in earlier sections, in this project, an Epoch is defined as "one cycle of all server-side processing". From a technical perspective, an Epoch is a PHP function that calls many other PHP functions in sequence to do all necessary data retrieval, processing, and API calls. This Epoch function is then called periodically via a scheduled Cron task.

To provide a clear understanding of how and when the EASIER system processes data, this chapter will describe all of the actions taken during the course of an Epoch. A graphical representation of an Epoch is found in Figure 2.4, however, this chapter aims to provide a more in-depth description of each step.

4.1 Find Active Devices

4.1.1 Step Description

In this step, the AWAIR API is sent an HTTP request to retrieve all registered AWAIR devices under the EASIER project's organisation ID. This step ensures that each registered AWAIR device is properly represented in the internal databases. Each device ID is stored in memory in a data structure to be used by the next step.

4.1.2 Step Inputs

The only data required to carry out this API call is the EASIER project's AWAIR Organisation ID, which does not change.

4.1.3 Step Outputs

After this step, a comprehensive list of device IDs associated with the EASIER project. These device IDs are stored in memory to be used by upcoming steps.

4.2 Fetch All Active Devices' Air Data

4.2.1 Step Description

The AWAIR API is once again queried over HTTP, and returns the most recent sensor measurement reported by each AWAIR air quality device. AWAIR returns the data as a JSON string, which is processed and, in combination with the previously retrieved device IDs, mapped to unique objects that are stored in memory for processing performed by upcoming steps.

4.2.2 Step Inputs

The AWAIR Organisation ID is required for the HTTP request. Data processed from this HTTP request gets associated with the device IDs that were already in memory.

4.2.3 Step Outputs

Now, each AWAIR device is mapped to an object also containing its most recent sensor fetch data. These parent objects can be thought of as a "home" for the context of an Epoch, and are stored in memory for use by future steps.

4.3 Fetch System-wide Heuristics

4.3.1 Step Description

Information regarding system-wide communication statuses is calculated and stored in memory. This step in effect is making a snapshot of the overall state of the system at the time of the Epoch, which is accomplished by looking at each home object. Specifically, measurements include: the number of non-responsive connected devices, total number of responses this Epoch, a list of all registered devices, a boolean indicating whether or not all devices responded this Epoch. This data is useful for visualisation of the system state as a whole at a glance, and can be viewed via the Homes Panel.

4.3.2 Step Inputs

System-wide Heuristics rely on comparing all home objects generated by prior steps. Additionally, to confirm "number of non-responsive" devices, these objects are compared with previous HTTP request results, to identify AWAIR sensors that were reported as existing but that did not report readings during this epoch.

4.3.3 Step Outputs

An object storing heuristic calculations is created and stored in memory, to be later inserted in a database.

4.4 Calculate Device Heuristics

4.4.1 Step Description

To better identify trends in measurements, and to simplify future data analysis, each Epoch's data is compared with the previous Epoch's data, to generate Device

Heuristics. Specifically, these comparisons record whether for any given device sensor's data between the previous Epoch and the current Epoch, whether the reading increased, decreased, or remained unchanged.

4.4.2 Step Inputs

A single database request retrieves last Epoch's sensor measurements and creates home objects that would be the same as those generated in last Epoch's "Fetch All Active Devices' Air Data" step. Then, last Epoch's home objects and the current Epoch's home objects are compared to generate heuristics.

4.4.3 Step Outputs

An object for the heuristics of each pair of home objects is created and stored in memory for use in future steps. These objects store the previous Epoch's and this Epoch's sensor readings for each sensor, as well as an enumerated flag describing whether or not the given sensor reading is higher or lower or equivalent this Epoch.

4.5 Fetch Alert Rules

4.5.1 Step Description

EASIER Alert Rules are the rules that define the thresholds for a sensor to generate an Alert in the system. Some of these thresholds are hard-coded in this step, but some are fetched from the per-device threshold database. In this step, an Alert Rule Object (Figure 3.1) is generated for each Alert Rule. An Alert Rule Object stores the enumeration for a specific sensor type, an operator (less than, greater than, equal to), and a threshold float value that is used for comparisons.

4.5.2 Step Inputs

The Alert Rule database is queried to return all threshold overrides, which will each become an Alert Rule Object.

4.5.3 Step Outputs

In addition to the Alert Rule Objects created from the Alert Rule database, an Alert Rule Object is created for hard-coded Alert Rules. These objects are stored in memory for use by future steps.

4.6 Process Alert Rules

4.6.1 Step Description

This step creates Alert Rule Holder Objects, one for each home object. The Alert Rule Holder Object is given a home object, and any number of Alert Rule Objects. In this step, a given Alert Rule Holder receives all Alert Rules associated with the home in the database, as well as one copy of each hard-coded Alert Rule. The Alert Rule Holder Object has an internal function that calls comparison operations supplied by each Alert Rule Object against the home object's associated sensor data. An "Alert" object is generated for each Alert Rule that is "violated". These Alert objects contain a reference to the home object that supplied the sensor data that led to the Alert being generated, a reference to the Alert Rule object that was violated, and the value that led to the violation.

4.6.2 Step Inputs

This step uses all home objects and all Alert Rule objects. A copy of each hard-coded Alert Rule Object is created to be used with each home object.

4.6.3 Step Outputs

An Alert Rule Holder Object is created for each home and stored in memory for use by future steps. These objects are used in this step to generate Alert objects (assuming some Alert Rules triggered an Alert to be generated). Each of these generated Alert objects is stored in memory for use in future steps.

4.7 Process Generated Alerts

4.7.1 Step Description

Currently, all Generated Alerts are currently only stored in memory and would be otherwise lost after this Epoch. In this step, an entry is created in the Generated Alerts database, one for each generated Alert. In doing this, the next time a given device queries the EASIER Central Server via the Android app, the server will be able to respond with all relevant information from the database, which is used to display in-app notifications of Alerts. Further, push notifications and SMS notifications are created at this step for each Alert, by sending an HTTP request to the Firebase and Twilio APIs.

4.7.2 Step Inputs

All generated Alert objects. Additionally, the database is queried for FCM tokens and phone numbers to send push and SMS notifications associated with Alerts.

4.7.3 Step Outputs

A new database entry will be created for any new Alert. If an Alert is generated of a type that already exists in the database, the existing entry will be either (a) updated to a higher "severity" level (indicating that an Alert has been generated multiple times with no acknowledgement from the user) if its status is "unacknowledged" or (b) set

from “acknowledged” to “unacknowledged” at the lowest “severity” level (indicating the same as a newly generated Alert, only via a database update instead of insertion.) Additionally, push notification and SMS notification API calls responses (typically only success/fail indicators) are also temporarily stored in memory.

4.8 Store Logs in Database

4.8.1 Step Description

Previously calculated system-wide heuristics and per-device heuristics are stored in the EASIER database for future analysis and comparison.

4.8.2 Step Inputs

Previously calculated system-wide heuristic objects and per-device heuristic objects.

4.8.3 Step Outputs

Each heuristic object is stored in the persistent EASIER database.

4.9 End of Epoch

At this point, the Epoch is complete. In this cycle, we have:

- Performed a heuristic check of the responsiveness of all devices in the EASIER system
- Retrieved most recent sensor readings for every device
- Compared these readings to previous readings
- Retrieved all Alert Rules
- Checked Alert Rules against the freshly retrieved sensor data

- Generated Alert objects
- Stored all of these calculated data in the persistent database
- Queried external APIs
- Stored Alerts in the database.

This means that the next time an Android device requests generated Alerts from the EASIER server, the database will indeed contain Alerts should they have been generated, allowing the app to display them to the user; thus fulfilling the core requirement of the EASIER project.

4.10 About Epochs

EASIER's Epoch based system allows frequent enough generation of alerts that the system seems, to the user, as "real-time" - even with the presence of a brief delay. This delay allows the server to be able to not get overwhelmed or rate-limited, and also allows users' (via the databases/Homes Panel or user devices) changes that may affect data calculation to propagate fully before they are used in calculations or comparisons.

After an Epoch is complete, after some time (currently 5 minutes), another Epoch will run to completion. Over the two year scope of this project, that amounts to 210,400 Epochs, each capable of generating tens to hundreds of database entries.

Chapter 5

Mobile Application

5.1 Developing with Flutter

Flutter was chosen to be the development platform for the mobile application portion of the EASIER project. Flutter is a UI software development kit that is developed by Google. Initially, Flutter was intended specifically for use in developing Android applications. Now, Flutter supports multi-platform executable compilation requiring minimal, or even no, code changes for publication to multiple operating systems. While the EASIER application is currently only used with one type of Android device, the ability to easily expand this in the future is a great benefit (Heimann and Veliz [2022]). Flutter applications are developed using the Dart programming language. Dart provides many features that make development incredibly easy (Patil et al. [2023]), such as the Pub package manager and software repositories. This enabled easy integration of external APIs like Firebase Cloud Messaging (used for push notifications in the project). One core design principle in Flutter is the concept of "widgets" - nearly everything in a Flutter application is a widget, and a widget may contain other widgets. This hierarchical, object based approach to application design is fairly unique. For this project, Flutter was selected in spite of its steep learning curve due to a desire to properly learn the framework for use in future projects.

5.2 Application Architecture

In describing the EASIER Android application, the term "Agent" will be frequently used. In Flutter, dynamic UI is laid out in code in an initial state. Its content (text, colour, images, etc) can be determined using changeable variables if the widget being used is a widget of the mutable StatefulWidget type. A variable in Flutter can be updated by the widget that uses it, or from other widgets/scripts running simultaneously (whether asynchronously or otherwise), at which point a setState call triggers a UI update with the updated value. Flutter attempts to increase app performance by only updating UI displaying values that have changed internally through a setState call. For the EASIER project, the scripts running asynchronously to communicate over the internet with servers and APIs (such as those used to fetch sensor data and receive alerts) are referred to as "Agents". These can be thought of as a sort of service - a standalone script that runs continuously (or over a specified interval), retrieving data and updating variables linked to UI as necessary. EASIER's Agents will be described below, and are shown in a chart format in Figure 5.1 as well.

What is "continuous"?

In the context of the EASIER Application, the terms "realtime" and "continuous" are used to refer to data and functions that update relatively quickly, generally with the intent of providing a user information that frequently changes. For example, the AWAIR sensor API may update the values it reports every 30 seconds, whereas the weather API updates the values every 2 minutes, and the EASIER Web Server updates the Generated Alerts database (as described in Section 3.3.2) every 5 minutes. All of these communications would be referred to as "realtime". Though they are not instantaneous, they provide information required for the application function frequently enough that any increased frequency would not improve overall functionality. For both the EASIER Application and the EASIER Web Server, changing actions' frequencies could affect performance should the EASIER project

continue to grow to a very large scale. However, the scope of the project is currently relatively small, processing required for all functions across a few dozen devices is performant enough that focusing on optimisation is less important than focusing on development of new features. The significance of "realtime" in general is not the number of seconds that have passed, but instead that the task is completed in time to accomplish its goal(s) successfully (Hu [2012]), which for the EASIER System is accomplished on the scale of minutes, not microseconds.

5.2.1 Agents

Push Notification Agent

The general purpose of the Push Notification Agent is to handle asynchronous communications with the Firebase API which handles EASIER's push notification delivery. Specifically, the agent passively checks for messages received from Firebase indicating the receipt of a push notification when the app is open. Delivery of push notifications when the app is closed is handled by Firebase's system-level activity, which is registered with Android via Firebase's API by the Push Notification Agent on the first launch of the app. The Agent initially requests push notification permissions, registers the system activity, generates an FCM token via the Firebase API, stores the token locally, and begins passively checking for messages from Firebase.

Web Alerts Agent

This agent is the most important Agent for the core functionality of the application, as it handles all Alert related communications with the EASIER Web Server. The Web Alerts Agent periodically sends an HTTP request to the EASIER Web Server to request any existing generated Alerts. It is also responsible for sending an HTTP request to indicate that the user has acknowledged an Alert and to mark it as read in the database. This agent also transmits the generated FCM token and locally stored device settings (used to pair a device/user with a specific AWAIR device ID)

for storage in the database. Lastly, the Web Alerts Agent is responsible for the processing of received data into a format that can be processed internally via Dart code, ultimately making it user readable. The following is an example of some data that might be processed by the Web Alerts Agent:

```
HUMIDITY:LESS_THAN:30:24.05:3810:1:1647:2024-02-25 03.10.03|CO2:GREATER_THAN
    :1200:1218:163:1:1677:2024-02-14 01.45.04|PM25:GREATER_THAN:15:16:187:1:1697:2024-02-24
    00.40.03|VOC:GREATER_THAN:500:833:192:1:1772:2024-02-24 23.45.04|
```

Above is an example of the resulting data from a single device's request for generated Alerts. This data is split and processed to be displayed to the user in the application.

AWAIR Agent

UI elements in the application that display realtime information from the user's AWAIR sensor receive data fetched directly from the AWAIR API via the AWAIR Agent, rather than receiving the data from the EASIER Web Server. This is so the device's refresh frequency is not limited by the server's Cron scheduling. This Agent makes requests to the AWAIR API and handles all processing of the raw data to a usable format, including formatting the data to a user readable string like the Web Alerts Agent does with EASIER generated data.

Lottie Agent

In-app animations for UI elements like weather conditions are generated via use of Lottie files. These files are fetched at runtime from a remote server by the Lottie Agent. The Lottie Agent maps internal widgets to URLs hosting the actual animation, allowing them to be easily swapped if necessary.

Preferences Agent

This agent handles persistent storage in the application by interfacing with Flutter's Shared Preferences framework. This is used to save things like API keys, the device ID, FCM tokens, recent fetched sensor data, etc. Shared Preferences uses asynchronous operations quite extensively, so there is a need to validate data on most read/write operations. This agent handles all interactions with reading and writing to Shared Preferences, condensing functionality throughout the rest of the application to an easy-to-use function call.

Weather Agent

In-app realtime weather data is periodically fetched from Open-Meteo's API via this Agent. The raw JSON data is also processed to a usable format and readable string as necessary via this agent.

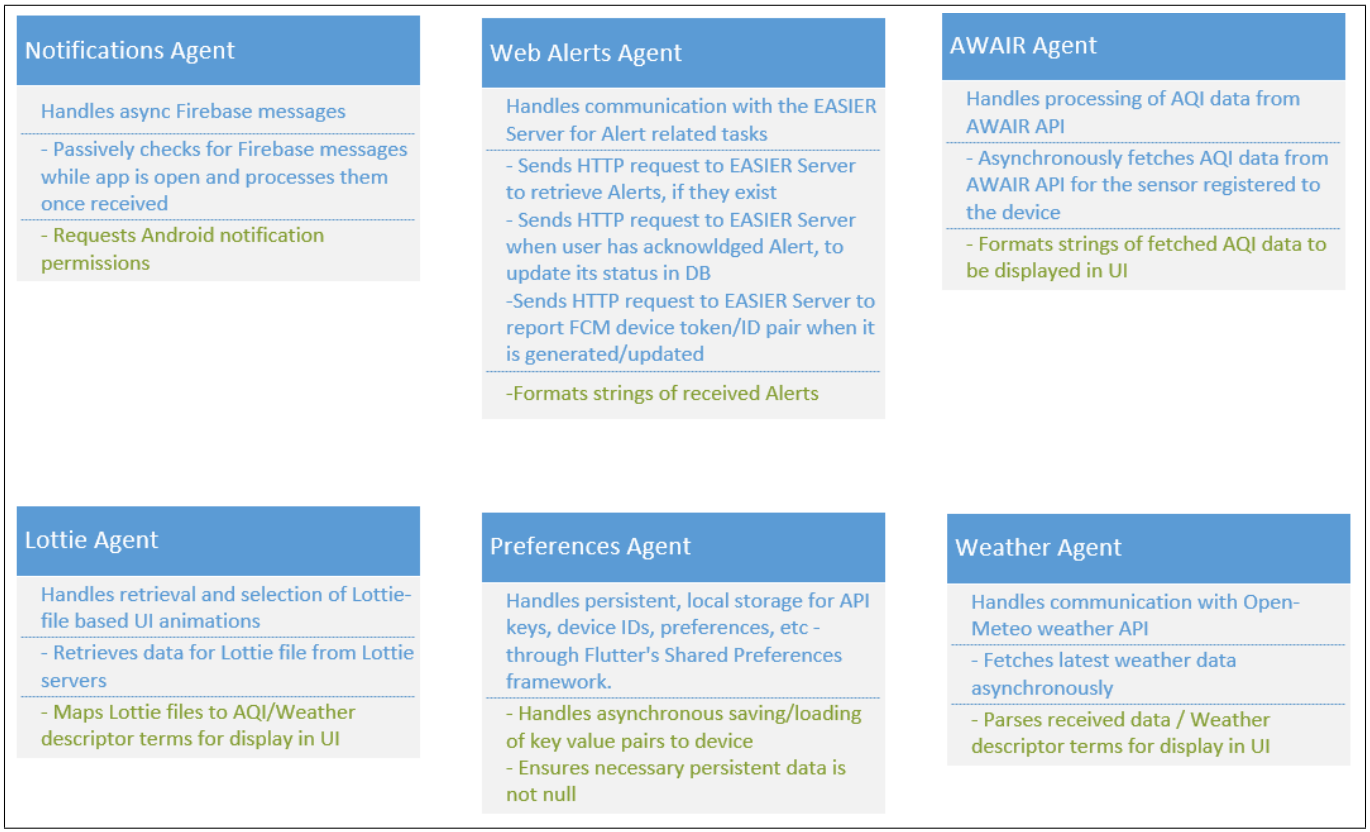

Figure 5.1: Application Agents Diagram

5.2.2 UI Flow

Flutter uses a special type of widget object called a "Navigator". This widget can be thought of as a complete system for handling navigation in an application, including navigating to and from subpages naturally, as well as handling tabbed interfaces, on-screen popups, and more. This was found to be sufficient for the needs of the EASIER application as its requirements for navigation are flexible and relatively uncomplicated.

In using the Navigator component, a number of distinct pages are made in separate classes and brought together by the Navigator element's UI flow control functionality. For the EASIER application, the following pages (which will be referred to as "Class" or "Page Class" henceforth) are as follows:

- HomeScreen (Figure 5.3.a)
- SensorInfo (multiple) (Figure 5.3.c)
- InfoMenu (Figure 5.3.d)
- Settings (multiple) (Figure 5.3.e)
- AlertScreen (Not shown)
- Alert Popup (Figure 5.3)

The pages labeled as (multiple) serve as an example of pages that have hardcoded variants. For example, there is a distinct CO2SensorInfo page class, HumiditySensorInfo page class, etc. Since these pages only display static information about the sensor, for the sake of simplicity they were hardcoded rather than being implemented dynamically.

When the application launches, it runs a *Main* class that initialises the Agents as described above, as well as the Navigator class that handles UI flow between the different pages. In The flow of the application's user interface can be best visualised

by looking at the UI flow chart, seen in Figure 5.2. Additionally, below is a more in-depth description of the flow's stages.

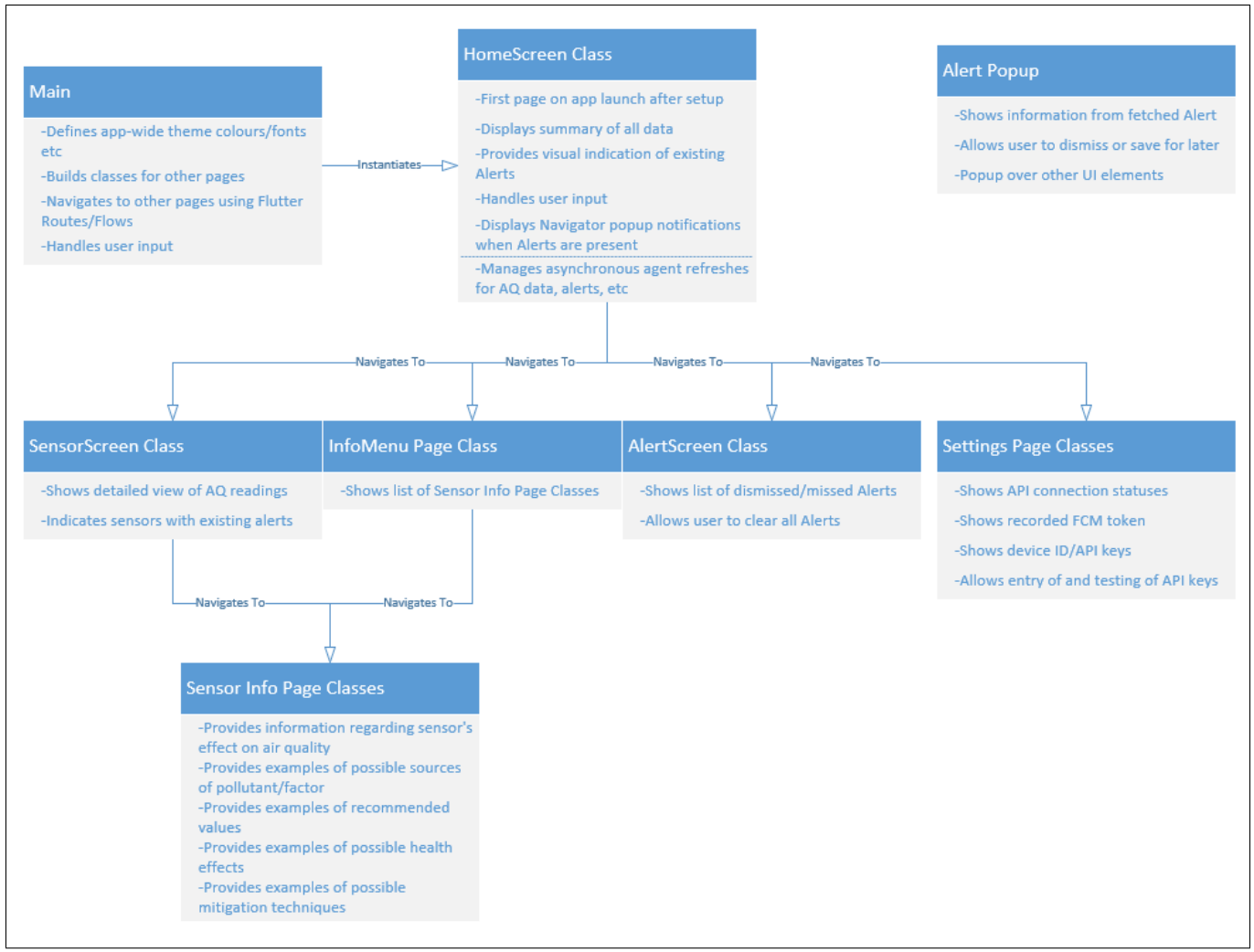

Figure 5.2: Application UI Flow Diagram

Main Class

This class is called on launch of the application. It defines system-wide theme colours/fonts/sizes and other theme definitions. It instantiates Flutter's built in Navigator class. It imports and builds the other page classes, and builds the flow by passing them to the Navigator. It also defines the toolbar on the top of the application, which contains buttons to quickly access the settings and the Alerts inbox, as well as the tabbed page selector on the bottom of the application.

This section also handles user input function definitions, such as handling Android's system "back" button, etc. This class also begins the Agents' execution, as described previously.

HomeScreen Class

The Home Screen is the default page upon application launch. If the application has not yet been setup (specifically, if there has not been an AWAIR device ID entered) it will automatically navigate to the settings page, with the intent of forcing the user to immediately set up the application. The following other elements are visible on the Home Screen.

- Summary Score - the summary AQI score as reported by the AWAIR devices. This value automatically updates. Its colour represents the quality level. When the Summary Score details button is clicked, the Navigator pushes the SensorScreen class.

- Outdoor Temperature/Humidity - as reported by OpenMeteo. These values automatically update.

- Air Threshold Alert - a visual indicator that alerts exist in the Alert Inbox. This was implemented in a recent update to reduce on-screen clutter and intrusive, frequent alerts.

SensorScreen Class

This section shows all current sensor readings from AWAIR in an easy to read format. Each sensor's name displays white by default, but changes to red to indicate that the current reading is generating an Alert by violating the Alert Rule in place. Each sensor's name also acts as a button to open the relevant InfoMenu Page.

SensorInfo Page Classes

There is one hardcoded InfoMenu Page Class for each sensor. Its goal is to inform the user of what the sensor is measuring (ie, "humidity is the measure of the amount of water vapor in the air."), how to prevent/mitigate it (ie, "check for leaks in plumbing"), possible sources (ie, "Hot Showers"), and other tips that serve to educate (fulfilling the Environmental Justice goal of this project) users on their home's air quality. These pages include pictures, interactive checklists, and other tools that may be helpful in an individual's understanding of their air quality.

InfoMenu Page Class

This section visually represents all of the SensorInfo Page Classes with a list of clickable icons including picture-based representations of the sensor's type. When clicked, these items navigate to the relevant SensorInfo Page Class. This page is accessed through the "info" Navigator tab.

AlertScreen Class

This page lists yet to be acknowledged alerts that have been dismissed or missed by the user. These alerts are stored locally using Flutter's SharedPreferences. The AlertScreen page was implemented to attempt to mitigate intrusively frequent popup alerts that interfered with the user experience.

Settings Page Classes

The Settings page is accessed either via the settings icon button in the Navigator toolbar, or automatically when the app is launched without a stored AWAIR sensor ID. The initial Settings Page displays the device's recorded FCM token (this is used in cases that it must be manually recorded by a volunteer when initial app setup or testing is being conducted) at the bottom of the screen, as well as a list of API-connection related buttons. Both of these API-connection buttons indicate the connection status with icons and colours. When clicked, they trigger the navigator to push an additional page (or series of pages) to progress through the setup process for the associated API. For the AWAIR API connection process, this simply navigates to a page that allows entry of the AWAIR device ID via the on-screen keyboard. If a device ID has already been entered, it will appear in the entry box automatically. A "connect" button will store the AWAIR device ID via SharedPreferences, then make a test API call (whose timestamp will be displayed for confirmation purposes immediately, also available later to verify recent connectivity) and allow the user to navigate back to the homescreen. A similar button exists to allow the user to begin the process of onboarding their device for Ecobee smart thermostat integration. Below this button is an additional "show ecobee data" toggle button that by default hides Ecobee related data throughout the application. Initially, in the early development of the EASIER project, it was the intention of the team to provide every home with an Ecobee smart thermostat, whose data would appear on the application and be used for alerts. However, due to difficulties regarding the complexity of installation in the homes of the users, only limited functionality was added to the application and the EASIER project as a whole. However, a few homes were actually equipped with an Ecobee thermostat, so some functionality had to be included in the application. Rather than creating separate branches of the application for this subset of users, it was decided to include a toggle that defaults to "off" to show the Ecobee functionality in the app.

Alert Popup

The Alert Popup class is pushed by the Navigator as a popup. It is displayed when an Alert is received while the application is open. The Alert Popup is accompanied by a notification sound when it appears. On the popup, a message displays a description of the relevant Alert's data, alongside a potential cause and suggested mitigation tips. For example: "The EASIER system has detected that your PM2.5 level rose above the threshold of 15 micrograms/cubic metre. Be mindful of indoor activities that release many particles in the air (burning candles, incense, etc.). If high PM2.5 levels persist, setting up an air purifier that contains a HEPA filter can help." There are buttons on each popup to navigate to a page with more information about the alert (one of the SensorInfo Page Classes) or to save the alert for later, moving it to the AlertScreen alert inbox page.

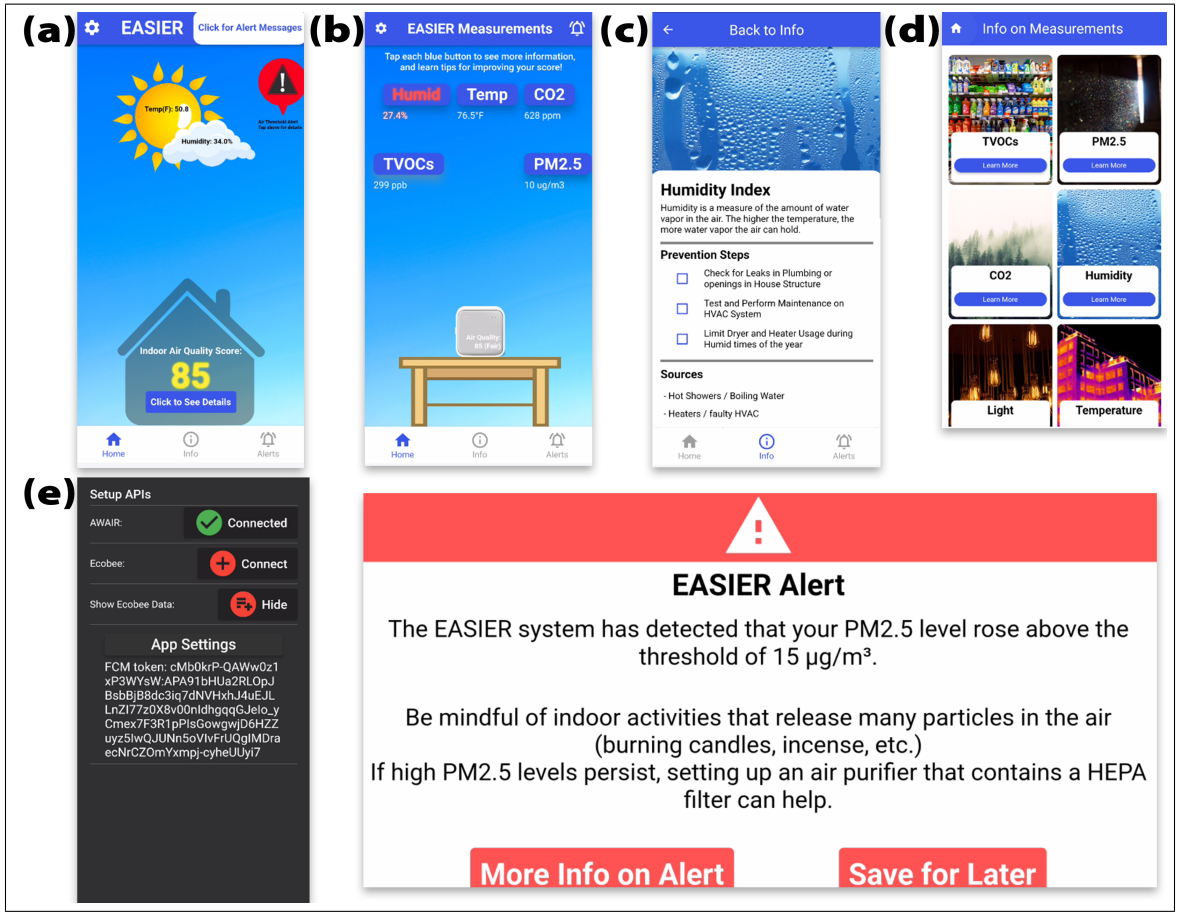

Figure 5.3: App Screenshots. (a): HomeScreen, (b): SensorScreen, (c): SensorInfo, (d): InfoMenu, (e): SettingsPage

5.3 Application Implementation

The interactions between the EASIER Agents, the UI flow, and the application as a whole are charted in Figure 5.4, though their relationship is very straightforward. All Agents are running asynchronously in the background, generating the values stored in variables mapped to UI elements (be they on-screen text, sounds, Lottie animations, images, or otherwise). The UI Flow functionality handles navigation between pages and on-screen data is updated automatically via Flutter's built in Stateful Widget functionality.

5.4 Application Distribution Flow

This section will briefly describe the methods of distribution of the EASIER application, its development, compilation, distribution, and maintenance.

5.4.1 Version Control

Development of the EASIER application was managed with a simple multi-branch Git workflow via GitHub. Various developers have been involved over the scope of this project, requiring lots of collaboration across many devices, so it should be no surprise that this process relies heavily on use of Git. A challenge is introduced in the development of the EASIER project as a whole when it comes to collaboration of development on a running web server that is currently in production being used by all of the system's devices. Since there is no built-in collaboration functionality on a standard shared web hosting platform like the one used for the EASIER project, some custom solution had to be implemented. There are many approaches to this, but since the EASIER project is relatively small and web/server development was handled mostly by one individual, version control was handled simply by integrating an automatic FTP transfer plugin in the development environment, automatically downloading/uploading changed server files on save/load,

in combination with another GitHub repository. Larger projects typically separate in-development and deployment versions of server-side code, allowing multiple branches to be developed simultaneously and pushed remotely to the web server when ready for deployment, but this was deemed unnecessary due to the scale of the project.

5.4.2 App Bundle Compilation

Flutter's built in compilation tools are used to compile app bundles to be distributed to the Google Play Store. App Bundles are a format allowing developers to upload multiple versions of an application, each targeting specific categories of devices, to the Google Play Store. In combination with Google's standard application security signing process performed via a few commands in a terminal, a ready-to-publish App Bundle file is created after each update is complete.

5.4.3 Google Play Store Publication

To make the process of distributing and updating the application across user devices easier, the EASIER project created a Google Play developer account for a small fee, allowing the application to be uploaded to the Google Play Store. This enables automatic updating of the application when a new version is pushed, as well as one-click downloads on the user devices during initial setup (bypassing the need for developing an in-house installation and updating method). Once an App Bundle is uploaded, Google conducts a manual review of the update's new contents, and within a few hours to a few days, the new version of the application is automatically installed on all users' devices. Google Play also offers analytics allowing us to ensure all users are running the proper version of the application among other data.

5.4.4 Installation to Device

When a new device is being set up, after a Google account has been created and associated with the device, the application is installed simply by searching the name of the application on the Google Play Store and clicking the “install” button.

5.4.5 Application Updating

All devices have Google Play’s “keep my apps up to date” feature enabled, meaning they automatically download and install new updates as soon as they become available. This mitigated the prior need to physically access each device and manually install a new update before Google Play was used for distribution.

5.4.6 Error Reporting and Statistics

In addition to the previously mentioned user version analytics, Google Play also provides information like crash reports, which can be helpful for bug fixes and versioning when developing the application.

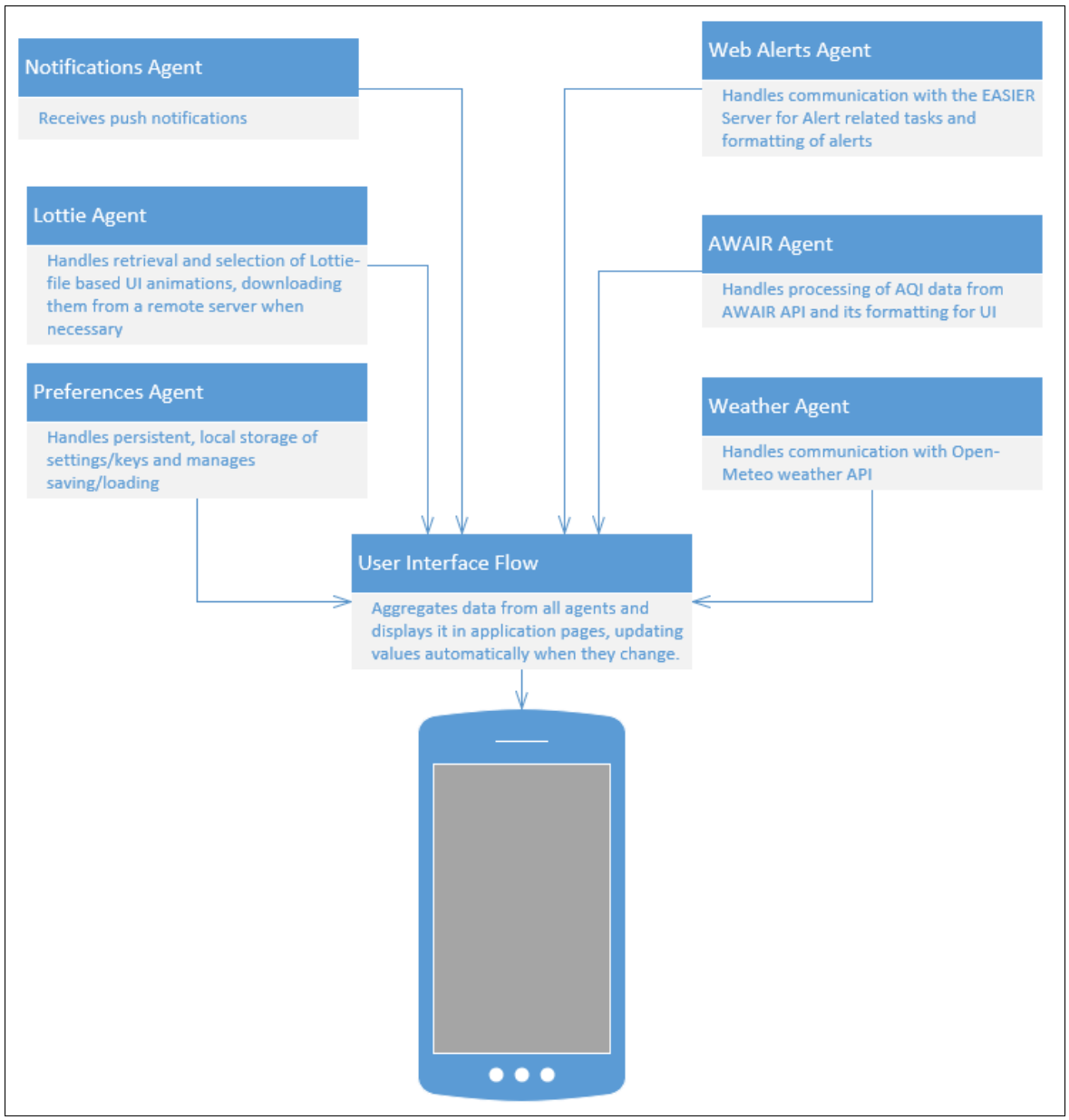

Figure 5.4: Application Agent and UI/functionality interaction chart

Chapter 6

Conclusions

6.1 Results

6.1.1 Sensor Readings

Sensor readings and heuristic data have been collected every five minutes since the initial implementation of EASIER's server code. The amount of data recorded for each epoch has steadily increased as more devices are onboarded. At the time of writing, over 1.5 million entries are stored in the sensor fetches database alone. This data is being compiled and analyzed with retrospect to provide insight to participants and their healthcare providers. Some of this analysis is as simple as providing a line graph with data points averaged across the weeks of recordings. However, more in-depth analysis is being conducted by the team with the goal of providing deeper insights in regard to home safety as a whole. With the recent proliferation of large language model based AI analytic tools, approaching the daunting task of processing such large amounts of data has become more accessible than ever. With sufficient analysis, the EASIER team hopes to identify correlation between measured data and external factors like uncontrollable regional outdoor air quality (that may be affected by factors such as weather, wildfires, or other natural disasters) or controllable factors like in-home renovation that may release pollutants in the air. There are efforts to find

connections between air quality and actual recorded health incidents of participants, as well.

Digital Archaeology for Air Quality

Through the data collected via EASIER, as well as other publicly available relevant data, we hope to use advanced techniques of analysis and visualisation to draw novel conclusions that have the potential to give the world a greater insight into the causes of and effects of air quality. With EASIER's large amount of collected data recorded by constantly monitoring the conditions of real homes, it is thought that valuable conclusions could be discovered in regards to protecting, improving, and maintaining one's own home air quality. This style of deep analysis on a very large, naturally generated dataset (especially if combined with other similarly naturally generated datasets) could be considered an extrapolation of the techniques of studying software development referred to as "Digital Archaeology". Dr. Audris Mockus, a pioneer of the concept of Digital Archaeology, describes in his research summary's description of a paper on the open-source Apache project (Mockus et al. [2000]), that Digital Archaeology is the study of "digital traces [that] reflect various projections of collective and individual activity... [that] represent a complex interplay among individuals, groups, their culture, and artifacts they produce". The goal of such study is to dive deeper than the simple collection and statistical analyses easily performed by standard data science techniques, instead interpreting results in a way that combines "disparate data sources... over extended periods of time to construct rich, detailed models of events and people, and developing analysis methods that can identify the evolution and causes of events".

In discussing air quality alerts produced by the EASIER system with participants, some anecdotal examples of this kind of complex cultural artifact not immediately evident via data alone have come to light, IE an incident that would otherwise be only represented as one spike amongst thousands of others. One participant's home consistently reported incredibly low air quality generating numerous alerts,

which led to an inquiry in to their home's conditions and found numerous devices in enclosed spaces producing large amounts of Ozone. Another participant revealed that they/their associates sometimes boil home cleaning solution on their stove as a makeshift deodoriser, which simultaneously releases large amounts of VOCs into their home. These insights would not be identifiable solely by the data measured by air quality sensors (though with hindsight, they would likely be recognised). However, through more in-depth analysis, in combination with other large datasets, as well as learning about the participants' communities and habits by communicating with them directly, the EASIER team can further environmental justice by providing insights into homes' air quality. This comprehensive approach to analysis of real, natural data can perhaps be thought of as "Digital Archaeology for Air Quality".

6.1.2 Project Success

The EASIER project has achieved all of its developmental goals, successfully deploying to 50 real homes in the community. The system has been operating functionally with no significant downtime, and has not experienced any major setbacks in development.

6.1.3 Project Reception

Participants have expressed much positivity in regards to their interaction with the EASIER project. Especially in comparison to early versions, the final user interface of the EASIER Android Application was found to be easier to use and more positively received in UX tests. This particular outcome is incredibly positive, as it is known that it is especially beneficial with older adults who may have reduced technological literacy, to collaboratively design usable interfaces via community informed user testing (Chiu et al. [2019]). Participants have shown engagement with the application and the EASIER project as a whole, expressing interest in furthering their understanding of what each air quality factor means and how it affects them -

which is the ultimate aim of providing "Environmental Justice". The EASIER project was selected to be presented as a poster at the 2023 CHASE ACM conference, allowing for connection with potential future collaborators with similar goals. There will also be an EASIER project workshop at the 2024 National Home Performance Conference.

6.2 Project Scope

Development Team

The team involved with development of the EASIER project grew to include multiple programmers, as well as dedicated UX testers and designers. Outside of the development team, dozens of others have been involved with the development of system requirements, installation in homes, device acquisition and setup, and user testing.

Requirement Changes

Over the course of this project, no significant structural changes needed to be made. Small changes like method of application distribution and the choice to omit inclusion of the Ecobee sensors across most homes occurred but luckily did not greatly affect development.

6.3 Future Work

6.3.1 Expanding Userbase

The sponsors of the EASIER project are pursuing the possibility of extending the scope to include residents of other communities outside Tennessee, as well as other communities in Tennessee. There is interest in collaborating with other organisations engaged in similar research.

6.3.2 Adding Functionality

Ecobee

There is interest in adding further Ecobee functionality to the EASIER system, such as allowing severe temperature alerts to trigger an automatic adjustment of the home's HVAC system as a safety precaution, amongst other suggested features.

Optimisation

To be fully scaleable, some work in regards to system optimisation would be necessary. For the relatively small scale during the scope of this project, this was not prioritised.

Data Summary

As the EASIER project reaches its end, features to automatically provide more detailed summaries to participants are being actively developed. These aim to provide extra benefit to the participants, and to act as a sort of "debrief".

Community Integration

As previously mentioned, there is interest in expanding the user's social network to include organisations like paramedics/fire departments, healthcare providers and local caregiving organisations to provide an extra level of support that can be automatically contacted when an Alert generates a social network communication.

6.4 Closing Thoughts

This thesis described the EASIER project's goals, its design requirements, its architecture, implementation, and results. In the end, this project successfully produced an air quality monitoring and alert system that functions via a mobile application, device package, and central server. The EASIER system is now providing empowerment through Environmental Justice via innovative techniques and novel

technology to 50 homes. Development successfully accomplished all sponsor-set requirements and implemented all feature specifications, providing a functional and expandable, stable system that can be used and evolved for the foreseeable future.

Bibliography

Adobe, 2024. URL https://business.adobe.com/blog/basics/push-notification-guide#:~:text=Pushnotificationscanbeone,increaseappengagementby%2088%25. 9

M. C. Chiu, P. H. Huang, and S. H. Tsao. Redesigning the user interface of a healthcare management system for the elderly with a systematic usability testing method. *Journal of Industrial and Production Engineering*, 36:324 – 334, 2019. doi: 10.1080/21681015.2019.1647883. 85

N. Choi and D. DiNitto. The digital divide among low-income homebound older adults: Internet use patterns, ehealth literacy, and attitudes toward computer/internet use. *Journal of Medical Internet Research*, 15:5 – 12, 2013. doi: 10.2196/jmir.2645. 3

Google, 2024. URL https://developers.google.com/learn/topics/flutter. 7

L. Heimann and O. Veliz. Mobile application development in flutter. *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education V. 2*, page 1199, 2022. doi: 10.1145/3478432.3499158. 65

T. Hu. Real time/zero time. *Discourse*, 34:163 – 184, 2012. doi: 10.13110/DISCOURSE.34.2-3.0163. 67

K. B. Mankad and P. Sajja. Utilization of web services for service oriented architecture. *Journal of Global Research in Computer Sciences*, 1:20–23, 2010. 11

Microsoft, 2016. URL `https://web.archive.org/web/20170707052149/https://msdn.microsoft.com/en-us/library/bb833022.aspx`. 11

A. Mockus, Roy Fielding, and James Herbsleb. A case study of open source software development: The apache server. pages 263–272, 02 2000. ISBN 1-58113-206-9. doi: 10.1109/ICSE.2000.870417. 84

V. Patil, D. B. Jagtap, S. Khodke, O. Jagdale, and A. Shitole. Flutter-modern and easy technology to build applications. *International Journal for Research in Applied Science and Engineering Technology*, 11 2:457 – 459, 2023. doi: 10.22214/ijraset.2023.49035. 65

B. Selic. *Unified Modeling Language (UML)*, pages 1–7. John Wiley & Sons, Ltd, 2008. ISBN 9780470050118. doi: https://doi.org/10.1002/9780470050118.ecse409. URL `https://onlinelibrary.wiley.com/doi/abs/10.1002/9780470050118.ecse409`. 12

G. Subrahmanion, D. Dalvi, and M. Tandel. Bootstrap and django framework. *International Journal of Advanced Research in Science, Communication and Technology*, 12 1:130 – 131, 2021. doi: 10.48175/ijarsct-2158. 33

R. N. Thomas. Introduction to the unified modeling language. *Proceedings. Technology of Object-Oriented Languages and Systems, TOOLS 25 (Cat. No.97TB100239)*, pages 354–354, 1997. doi: 10.1109/TOOLS.1997.681883. 12

TVA, 2024. URL `https://www.tva.com/energy/technology-innovation/connected-communities/connected-communities-pilots`. iv

B. Weiss, R. Reed, and E. Kligman. Literacy skills and communication methods of low-income older persons. *Patient education and counseling*, 25 2:109–19, 1995. doi: 10.1016/0738-3991(95)00710-H. 1

Vita

Bryceton Bible is a Computer Science student at the University of Tennessee. He is a graduate research assistant in Dr. Xiaopeng Zhao's lab, focusing on research with AI, robots, and Virtual Reality as assistive and educational devices for People Living with Dementia and their caregivers. Bryceton loves language learning, and speaks, teaches, and studies Japanese and Esperanto, among other languages. Bryceton has worked to develop web based tools for Kansai Gaidai Foreign Language University in Japan during his time studying with KGU. Through his research, Bryceton hopes he can strengthen connections across the world.