MQP CR – 1502

SlipBuddy: A Machine Learning based Mobile Health App to Track and Prevent Overeating

A Major Qualifying Project
Submitted to the Faculty
of the

WORCESTER POLYTECHNIC INSTITUTE

In partial fulfillment of the requirements for the
Degree of Bachelor of Science

By

Joseph Acheson, CS
Joshua Allard, CS
Andrew Busch, CS
Andrew Roskuski, CS

2015-2016

Sponsored by:

Sherry L Pagoto, Ph.D, University of Massachusetts Medical School
Jessica Oleski, University of Massachusetts Medical School

Approved by:

Professor Carolina Ruiz, Advisor
Professor Bengisu Tulu, Co-Advisor

Abstract

We designed, developed and tested SlipBuddy—an Android application that collects data about overeating episodes, identifies contextual patterns around overeating episodes and provides interventions when users are likely to overeat. SlipBuddy was piloted with a small group of users from the community.

Table of Contents

1 Introduction

According to reports from the National Institute of Health [1], in 2010, more than two out of every three adults in the United States were considered to be obese, and as such, weight loss therapies are a serious area of research. One of the most popular techniques for weight loss is behavior therapy [2]. In addition to adjustments in food intake and physical activity, strategies including self-monitoring, stress management, and stimulus control are important parts of losing weight. Unlike dietary restrictions and exercise regimens, however, one's behavior is a constantly changing environment that must be watched in order to keep it under control.

We propose that a convenient and useful tool to help support this type of weight loss therapy is technology. Constantly being aware of one's eating habits and stress levels is hard to do, and poor understanding of oneself can lead to overeating. Instead, that awareness could be managed with help from a patient's smartphone.

Since many people have a smartphone on them at all times, it makes perfect sense to start using this personal device for a wider range of things, including health care. With devices like the FitBit, companies have started to look at the smartphone not just as a phone but as a device that is capable of logging and working with data in a completely new way. We believe that the mobile phone space is a revolutionary space that will make an extraordinary impact in health care in the years to come.

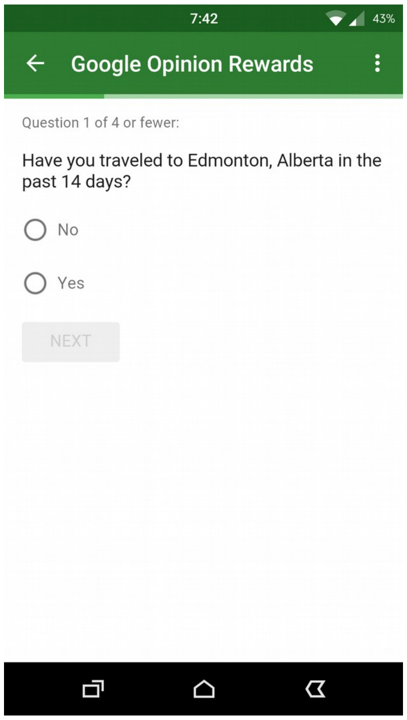
The goal of our project was to design, create and test a smartphone application that will help people who struggle with overeating be more aware of and in control of their eating behaviors. Specifically, this application records a patient's overeating episodes, along with other factors such as stress level and location, and creates a personalized report of the patient's behavior in order to intervene in any future overeating.

The interventions provided by SlipBuddy are intended to recognize and report on patterns in users’ behaviors that the users themselves may not know about. It will do so by taking into account many factors of a user’s behavior, and then personalizing the interventions for each user through the use of machine learning algorithms. We hope that, by predicting overeating episodes and then intervening before the users overeat, their awareness of their dietary patterns and their overall health will improve.

The next chapter will provide background information about research findings and applications that are relevant for our project. Chapter three will explain the implementation of our application—SlipBuddy—and the details of its system architecture. Finally, chapter four will present our conclusions and recommendations.

2 Background

SlipBuddy is a survey-centric application that uses data submitted by users to create interventions that will positively influence user's eating habits. The initial part of SlipBuddy involved creating an easy and comfortable survey experience for users. To do this, we looked at how other apps surveyed users. Applications like Google Opinion Rewards [3], which offer users small amounts of money for completing surveys, use several clear design choices to make getting responses to surveys clear and easy for prospective users. After beginning a survey, Google Opinion Rewards presents only one question to the user at a time, and they are only presented with the next question when they have answered the one presented to them. As illustrated in **Figure 2.1**, the presentation of each question is plain to ensure minimal misinterpretation of individuals in the user base. The user is only given the option to move on when they have entered a value. There are a number of other mobile survey applications that have adopted a similar format of question presentation, with variations depending on the specific goals of the application.



Question 1 of 4 or fewer:

Have you traveled to Edmonton, Alberta in the past 14 days?

No

Yes

Figure 2.1: Google Opinion Rewards

There is an unprecedented number of apps that claim to offer effective weight loss plans but fall short on using proven methods. A study on weight loss apps for both Android and iOS found that most apps use, on average, just under 4 of 20 behavioral strategies derived from an evidence based weight loss program called The Diabetes Prevention Program (DPP). [4] The study assessed a selection from the top 100 paid and top 100 free apps from both Android and iOS. These apps were put through an assessment phase during which the apps were reviewed for their usage of the 20 behavioral strategies employed in the DPP. The results were discouraging. Essentially there is a vacancy of weight loss apps on app stores that make full use of proven and effective weight loss strategies. [4]

Many studies are exploring the lack of effective weight loss apps available. One study attempted creating a solid integration of known effective methods with modern technology. The study looked to adapt the DPP, a system whose cost and difficulty of implementation has prevented its widespread use, into a more scalable electronic model by reducing the amount of in-person contact that the program recommends. Replacing or reducing the amount of in-person contact would vastly increase the use of an effective system like DPP as great cost comes with each in-person session, and that cost cannot stand to be scaled. [5] To accomplish this large scale alternative to in-person meetings, researchers created the ENGAGED app, which used smart phones as the users' window to the application. The study will compare their new weight loss app against other existing weight loss applications to see if there were significant improvements. [5]

Smartphones provide a new mechanism that can be used to positively influence the lives of users. Being effective in influencing users is a massive challenge in itself. In *Psychology & Health*, Susan Michie outlines 40 techniques and heuristics for causing effective behavior change. [6] Many of these techniques discuss what information should be presented to the user, when and how often the information should be presented, and how the information should be presented. For example, one technique says users should be prompted to review the outcome of previous goals they had set. Generally this information should be shown after the completion, or attempted completion of a goal.

A very generalized summary of the techniques described follows a few clear steps. First, ensure that the user understands their current circumstances, consequences of their actions, and a general idea of the normative actions of others. Next the user is prompted to set goals and plan out future steps. A large number of techniques address the most important part of the process which is assisting the user with working toward and achieving goals with instructions, information, and feedback. After achieving goals, the user is provided with more information on how to change and then reviews current goals as well as modifying those goals or making new goals. Throughout the whole process there must be elements in place to prevent the user from falling back and relapsing. These heuristics and techniques play a large role in the second phase of SlipBuddy, which attempts to influence and curb undesired behavior in users. [6]

Creating a system that persuades users to change their behavior is no simple task. One group of researchers, Harri Oinas-Kukkonen and Marja Harjumaa, have looked into creating effective persuasive systems built for influencing people’s behavior. [7] Their findings for effective system components in a persuasive system are very powerful. The concepts start with reduction of ideas. Keeping information simple and understandable is crucial. Oinas-Kukkonen and Harjumaa say a lot about tailoring an experience to an individual user, and meeting their needs. Among the most important features of a persuasive system are rewards. Rewards are also very important to the process, as they help ensure users will come back to the system and work for more praise. Another crucial element is that users must trust the persuasive system, and view it as an ally, rather than an opponent. [7] These ideas are very useful for persuading users to make healthier choices and sticking with valuable programs.

3 Development of SlipBuddy

This section discusses how we went about taking the initial concept and implementing the SlipBuddy application. We discuss both conceptual and technical implementation details. The implementation is presented in the form of two major phases. In the first phase, we describe how we built an application to facilitate regular data collection from users as easily as possible. In the second phase, we describe how we built a system that involved using the data collected from phase 1 to help users gain better understanding and control of unhealthy eating habits. **Figure 3.1** shows our development approach and system architecture design presenting at the high level how the different components of the system communicate with each other.

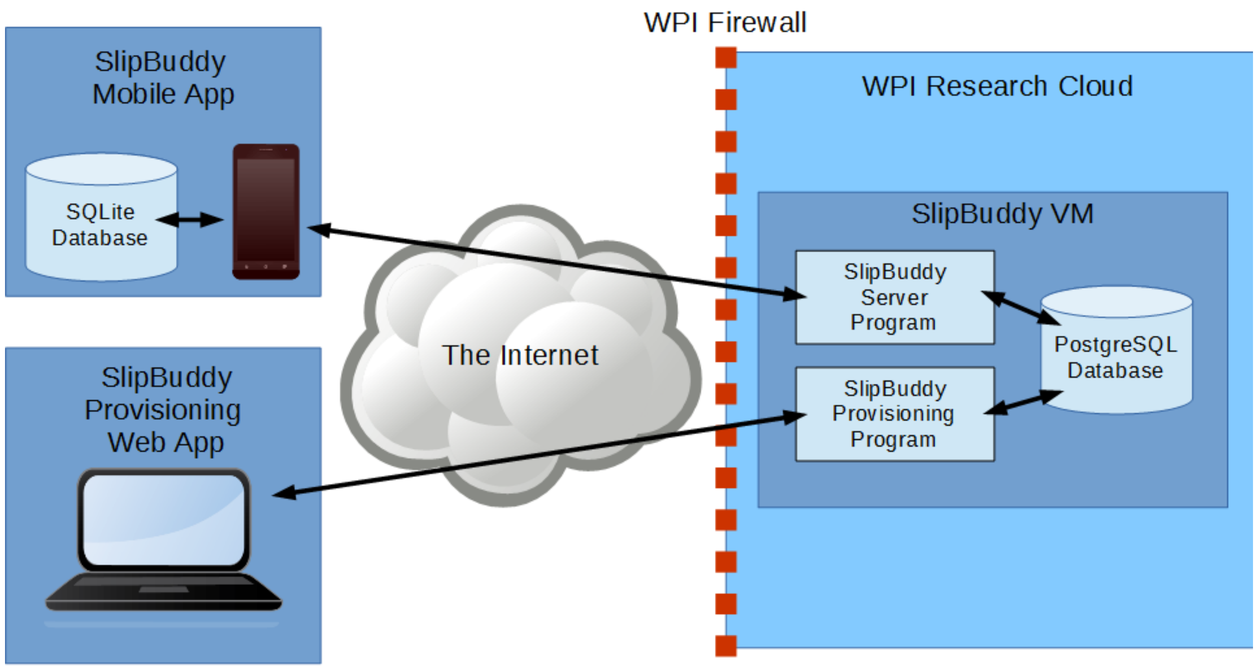


Figure 3.1: Component Communication Diagram

Interchangeable Terminology:

Before proceeding into the development of SlipBuddy, it is important to understand the terminology used throughout this report, the application code, and the finished products. It is also important to note that some of these terms have different aliases that are used interchangeably; these aliases resulted from a difference between our initial assumptions and requirements and our final design decisions for the user interfaces. **Table 3.1** lists the interchangeable words we used, and a brief definition.

Term & Aliases	**Definition**
User, Subject, Patient	This is a person physically using the SlipBuddy Android App.
Slip, Episode, Overeating Episode	This is the event recorded when a user feels that they have overeaten.
Check-in, Question, Status Update	This is the information the user would fill out at 3 preset times each day, asking about stress, hunger, sleep-quality, sleep-hours, and/or weight.
Sleep	This is the user estimated value of how well they slept over the past night.
Doing, Activity	This is the situation the user was in when experiencing an overeating episode.
Eating, Meal	This is what the user ate during an overeating episode.
Check-in-type, QType	This is the time of day for a check-in, either being ‘MORNING’, ‘AFTERNOON’, or ‘EVENING’.
Phone-PKID, ID	This is the value used as primary keys on the user’s phone for the different tables. This value is stored on the server, but useless since each user would have conflicting ids for their values. **Note:** ‘ID’ is also used in other situations as well, such as for User IDs.

Table 3.1: Interchangeable Terminology

3.1 Phase 1

The goal of Phase 1 was to create a data collection application. We sought to make a place where users could easily enter data three times daily without headache. We had the additional goal of a less constrained method of data input, where users could report that they had overeaten, and given information about the overeating episode they experienced. This section will detail our specific goals for the application, and our development of the application, both conceptually and technically.

3.1.1 SlipBuddy App Requirements

The development of SlipBuddy revolved around adhering to the requirements received at the onset of the project, with a few additions created as a result of further research and discussion. **Table 3.1.1.1** presents the requirements we gathered.

Requirement	**Requirement Description**
User can record an overeating episode	● User can enter location if "New", Map page opens. ● User can enter a time, which defaults to the current time ● User can choose what they were doing from pre-selected options ● User can enter detailed information about what they were doing ● User can choose what they were eating from pre-selected options ● User can enter detailed information about what they ate ● User can enter a stress level (0 = not stressed at all; 10 extremely stressed) ● User can enter a hunger level (0 = extremely hungry, 5 = Comfortably full, 10 = uncomfortably full)
User is prompted 3 times a day for a status update	● Morning update (9am): Questions 1, 2, 3, 4 ● Afternoon update (2pm): Question 1 ● Night update (8pm): Question 1 ○ If there are no reports for the given day, Question

Requirement (Cont.)	**Requirement Description**
Question 1, Stress Question:	● Three times a day (9am, 2pm, 8pm) the app will ask "Hello! What is your stress level on a scale of 0 to 10 (0 = not stressed at all; 10 extremely stressed)?"
Question 2, Hunger Question:	● Every day at 9am (default value – triggered with the first stress question) the app will ask "On a scale of 0 to 10, How hungry are you right now? (0 = extremely hungry, 5 = comfortably full, 10 = uncomfortably full)"
Question 3, Sleep Question:	● Everyday at 9am (default value – triggered with the first stress question) the app will ask "How many hours did you sleep last night?" and "How well did you sleep last night? (1 = very bad; 5 = excellent)"
Question 4, Weight Question:	● Every day at 9am (triggered with the first stress question) the app will ask "What is your weight today?" and will allow number entries (including decimals)
Question 5, Entry reminder question:	● If by 8pm no overeating episodes are recorded, the app will ask at 8pm (with the stress question) "You didn't enter any overeating today. Was there an overeating episode you forgot to enter?" (Yes/No)" ■ Will open the Overeating Pages if "Yes"
Don't allow users to have the same location label for two different locations	● For example, if they move to a new home, they will need to make a new home location
Status updates should be differentiated by time of day	● Morning, Afternoon, Evening

Table 3.1.1.1: System Requirements for SlipBuddy App

3.1.2 Development Platform Decision

Initially, we planned to build apps for Android and iOS platforms and structure the code so that the majority of it could be shared between platforms. We investigated various methods and toolkits for accomplishing this, as well as generally investigating and experimenting with the development tools for both platforms. Eventually, we settled on building native, platform specific

UIs while writing code that could be shared between platforms in C++, since that language was able to integrate with both platforms. We made an attempt to start developing the app this way, but quickly found that this approach was vastly more challenging than we had anticipated. After some deliberation, we decided that writing completely separate, native versions of the app for each platform was the best approach, because integrating native code was more difficult than anticipated, the native code portions were likely to decrease the overall code quality of the app, and the amount of code that could be shared between platforms was far less than anticipated, meaning that we would be putting in a lot of work to share a very small amount of code. Since the Android version of the app would be vastly easier to roll out to end users, that version of the app was determined to be the main priority.

3.1.3 SlipBuddy App Back End Design

For the first phase implementation, we developed an app to store data on the user’s phone. To this end, we used Android’s built in SQLite implementation, since despite its limitations, it was sufficient for the data we intended to store.

A simple database schema is sufficient to capture the data collected from status updates and overeating episodes, as well as the participant ID and the list of locations to which the user assigned a name. All of this data is stored in an SQLite database on the user's phone. The database file is stored in the app’s private storage, which cannot be accessed by other apps without root access. A mechanism we put in place to allows the file to be copied to the user-visible file system, so that it can be manually collected. This mechanism was put behind a hard-coded password in order to prevent it from being accidentally triggered by the user. There were 3 main database tables. The 'location' table, described by **Table 3.1.3.1**, stored the locations the user registered for reporting episodes. The 'episodes' table, described by **Table 3.1.3.2**, stored all the data associated with the episodes the user entered. The 'questions' table, described by **Table 3.1.3.3**, stored all data associated with status updates entered by the user. Additionaly, a 'subjectid' table, described by **Table 3.1.3.4**, was created so that the database file could be identified after being exported from the user's phone. The schemas used for the tables are as follows:

Column Name	**Type**	**Description**
name	text	Name of location and primary key
latitude	real	Latitude of the location
longitude	real	Longitude of the location

Table 3.1.3.1: Table 'location' Schema

Column Name	**Type**	**Description**
id	integer	Primary key
timeoccur	datetime	The time the episode occurred
timestamp	datetime	The time the episode was reported
doing	text	What the user was doing when the episode occurred
doingDetail	text	Detailed text about what the user was doing when the episode occurred
eating	text	What the user was eating when the episode occurred
eatingDetail	text	Detailed text about what the user was eating when the episode occurred
stress	integer	Stress level reported for the episode
hunger	integer	Hunger level reported for the episode
location	text	The name of the location where the episode took place

Table 3.1.3.2: Table 'episodes' Schema

Column Name	Type	Description
id	integer	Primary key
timestamp	datetime	The time when the check-in was recorded
date	date	The date of the check-in
qtype	text	Which of the 3 daily check-ins this entry corresponds to
stress	integer	The stress level reported during the check-in
hunger	integer	The hunger level reported during the check-in, if applicable
sleep	integer	The sleep quality level reported during the check-in, if applicable
sleepHours	integer	The number of hours of sleep reported during the check-in, if applicable
weight	real	Weight reported during the check-in, if applicable

Table 3.1.3.3: Table ‘questions’ Schema

Column Name	Type	Description
id	text	The ID of the user

Table 3.1.3.4: Table ‘subjectid’ Schema

In order to easily represent the collected data in memory, several Java classes and enums were created. The StoredLocation, Episode, and Question classes were created to mimic the database schema, while the Eating, Doing, and QType enums define the valid values for the corresponding fields. To simplify database operations, a database class was created which could both break down the Java objects and store them in the database, as well as retrieve them from the database when necessary. Some other methods, which reported certain statistics from the database, such as whether status updates had been recorded or getting how many episodes were recorded, were also created.

3.1.4 SlipBuddy App Front End & UI Design

Design Decisions

In order to understand the final application, it is important to discuss some design decisions that pertain to many of the pages. The first noteworthy decision was to have all number entries on SlipBuddy displayed using number pickers on the page, as opposed to number text entries; we made this design decision in order to keep the most control over the number inputs while still providing clear instructions to the user. For example, as illustrated in **Figure 3.1.4.1,** the use of number picker makes it clear to the user that they can only enter a four digit number, while **Figure 3.1.4.2** shows the same property with a one digit number. We also believed the number picker to be more visually appealing than a simple number text field. The application uses a slightly modified version of the default Android number picker, because it only shows three numbers vertically, which previous studies showed might be confusing to normal study participants; with the default number picker, users were not aware that there were more numbers hidden beyond the visible 3, which we hoped using five numbers would fix.

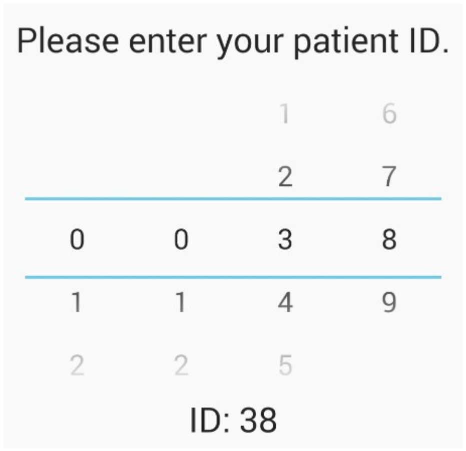


Figure 3.1.4.1: 4 Digit Entry

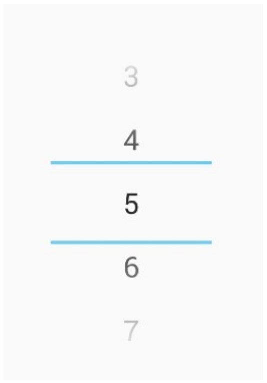


Figure 3.1.4.2: 1 Digit Entry

Project Overview

Figure 3.1.4.3 shows an overview of our app design. Like all other Android projects, SlipBuddy starts with the **Main Activity**. This is the first activity that is loaded when the app is opened. There are three components to the Main Activity: the **Tab Controller**, the **Navigation Drawer**, and the **Notification Manager**.

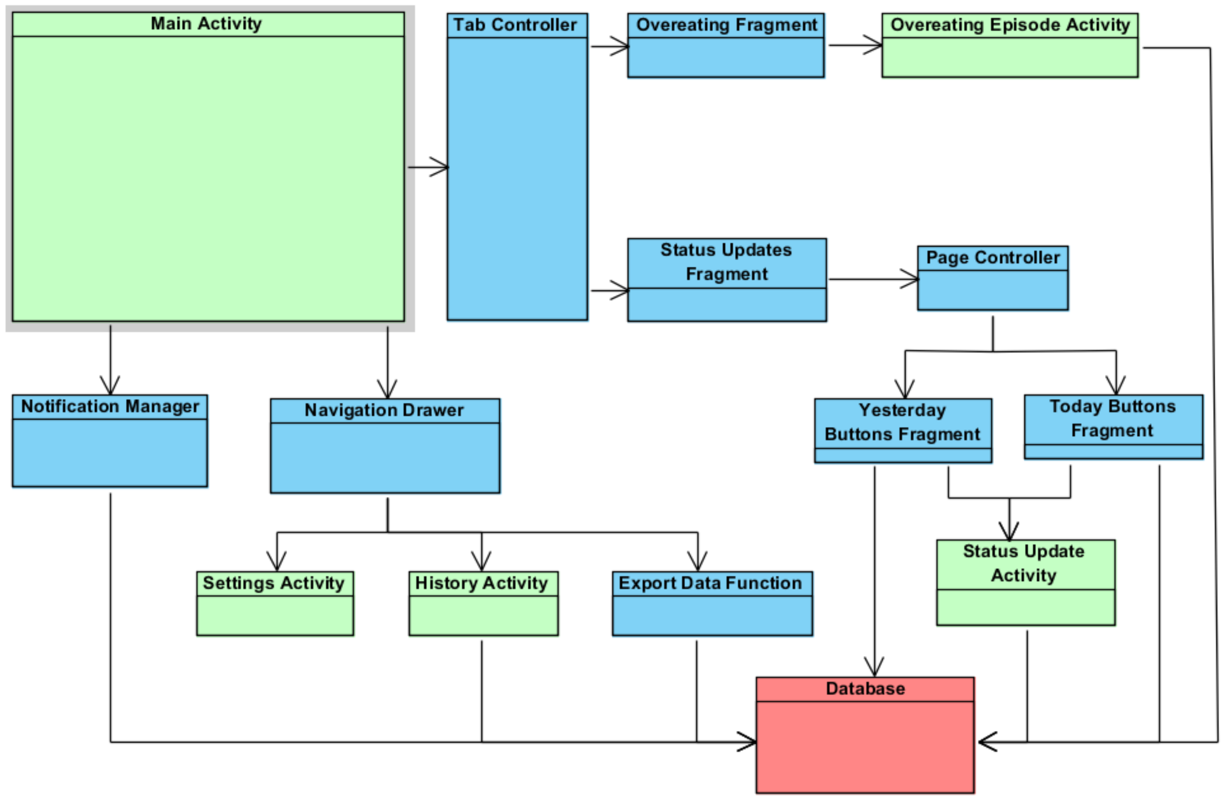


Figure 3.1.4.3: App Interaction Layout

The Tab Controller divides the two main data entry portions of the app and controls the tabbed view of the Main Activity. The left tab is the **Overeating Fragment**, which opens the **Overeating Episode Activity** for entering overeating episodes, which are then stored in the **Database**. The left tab is the **Status Updates Fragment**, which controls the status update functions. The Status Updates Fragment has an instance of the **Page Controller**. The Page Controller allows for swapping between its fragments, the **Yesterday Buttons Fragment** and the **Today Buttons Fragment**, with next and previous buttons. Upon selecting any of the incomplete status update buttons within these fragments, the **Status Update Activity** will open to allow the user to create a status update. These status updates are stored by the Activity into the Database. If the selected status update has been completed, the Activity will access the Database and display the completed status update.

The Navigation Drawer shows the settings for the application. The Navigation Drawer contains the **Settings Activity**, the **History Activity**, and the **Export Data Function**. When clicked on, these are launched by the Main Activity. The Settings Activity contains the settings that can be altered by the user. The settings are defined in "res/xml/pref_general.xml". The History Activity displays a scroll of all of the overeating episodes that have previously been entered. The Export Data Function exports the database object to a local directory.

The Notification Manager handles the notifications that the app displays. It accesses the Database to get the stored settings.

New Episode Overview

Figure 3.1.4.4 shows how a user enters an overeating episode. A **New Episode** has 5 different fragments, that are in a specific order but can be navigated freely by the user. The fragments are controlled by an instance of the Page Controller.

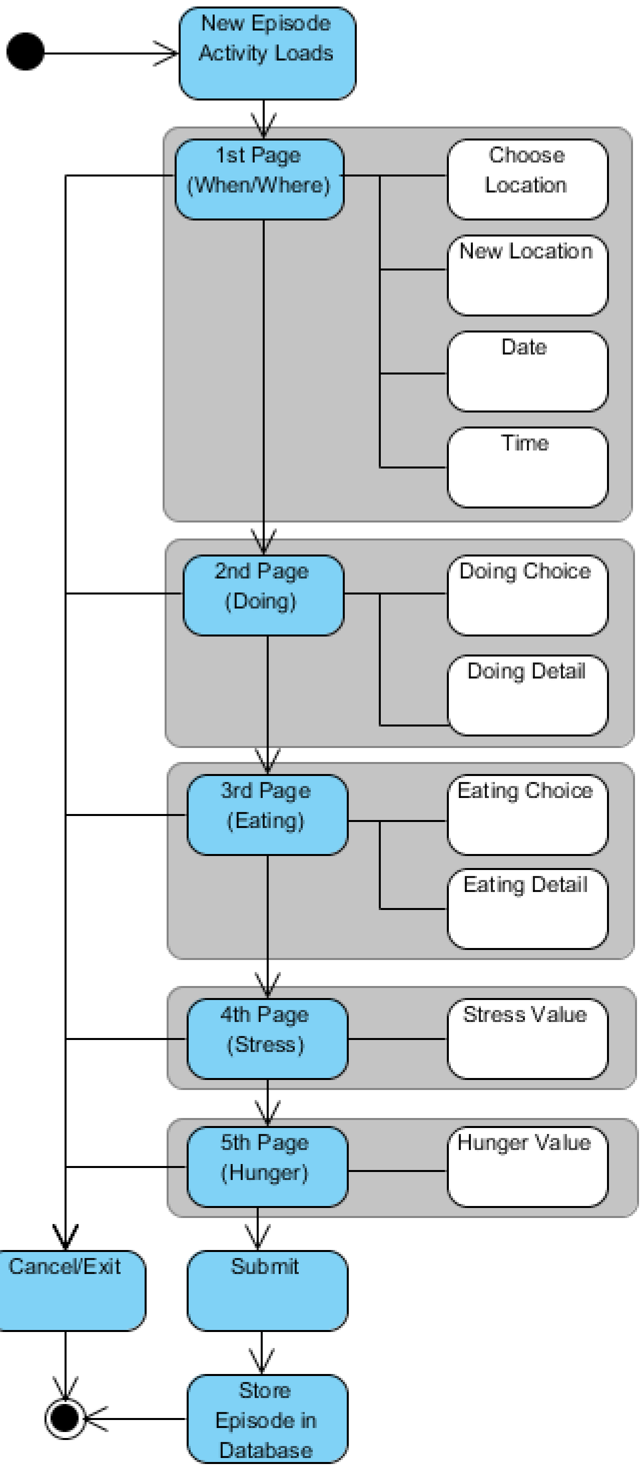


Figure 3.1.4.4: Recording Overeating Episodes

The first fragment is the **Location Fragment**. This fragment has requires a GPS location, a date, and a time. A GPS location can be entered via the Google Maps entry page, which requires a name. GPS coordinates can be given from the phone, but are optional.

The second fragment is the **Doing Fragment**. This fragment requires a dropdown Doing choice and an optional detailed Doing text fill-in.

The third fragment is the **Eating Fragment**. This fragment requires a dropdown Eating choice and an optional detailed Eating fill-in.

The fourth fragment is the **Stress Fragment**. This fragment requires a NumberPicker value of stress from 0-10.

The fifth fragment is the **Hunger Fragment**. This fragment requires a NumberPicker value of weight from 0-999.

The episode can be submitted on the fifth page after all required fields have been filled out; the activity performs a check. A new episode can also be canceled at any time. Once submitted, the episode gets stored in the local database.

Status Update Overview

Figure 3.1.4.5 shows how a user enters a status update. A **Status Update** has 6 different fragments, but which ones are shown are dependent on the time of day. The fragments can be navigated freely by the user. The fragments are controlled by an instance of the Page Controller.

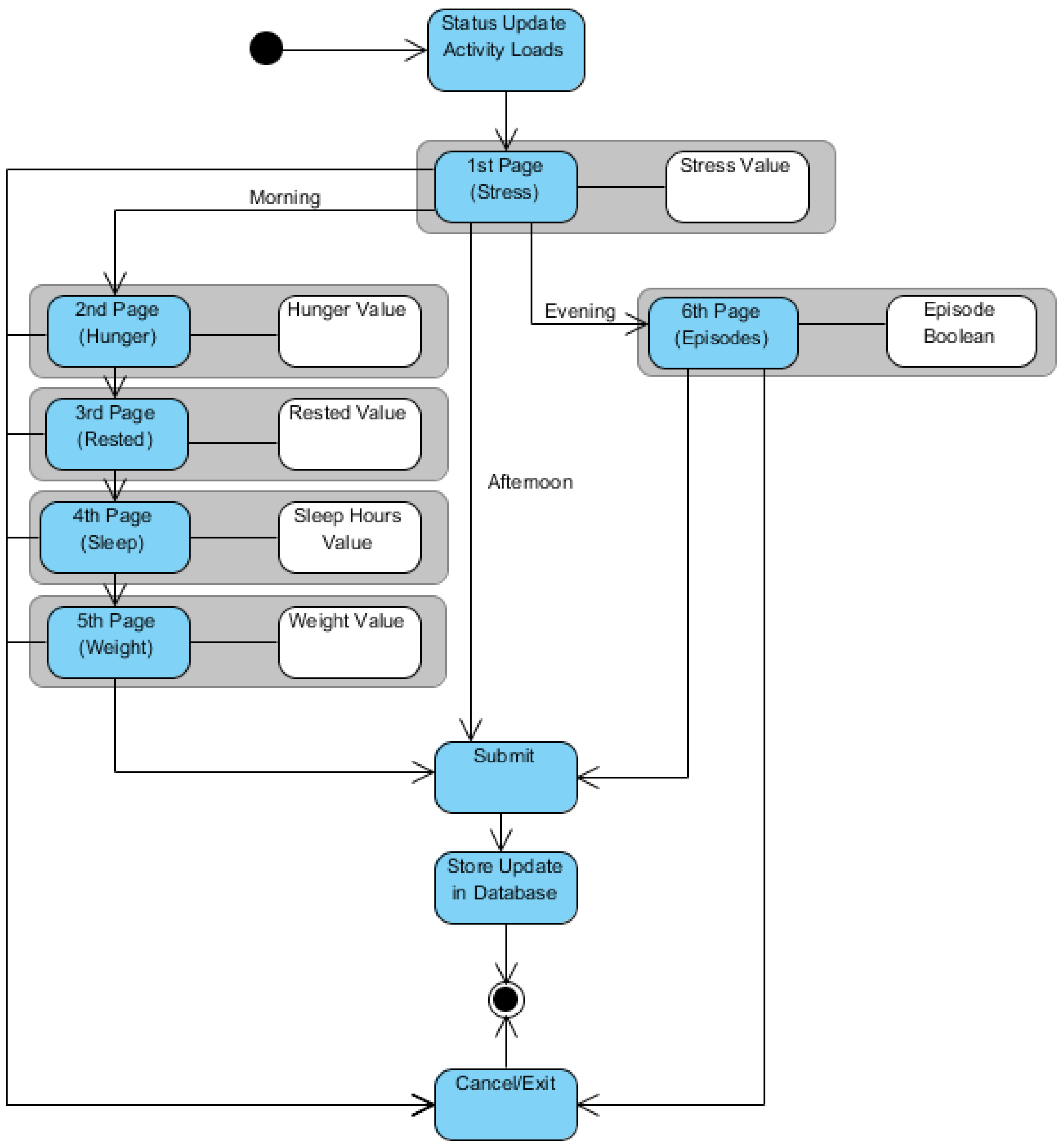


Figure 3.1.4.5: Recording Status Updates

The first fragment is the **Stress Fragment**. This fragment requires a NumberPicker value of stress from 0-10.

The second fragment is the **Hunger Fragment**. This fragment requires a NumberPicker value of hunger from 0-10.

The third fragment is the **Rested Fragment**. This fragment requires a NumberPicker value of restedness from 0-10.

The fourth fragment is the **Sleep Hours Fragment**. This fragment requires a NumberPicker value of hours of sleep from 0-24.

The fifth fragment is the **Weight Fragment**. This fragment requires a NumberPicker value of weight from 0-999.

The sixth fragment is the **Episode Boolean Fragment**. This fragment asks the user if they forgot to enter any overeating episodes for the day. If yes, the New Episode activity will open after submission.

The fragments that are shown depend on the time of day. The **Morning** update shows fragments 1-5. The **Afternoon** update shows fragment 1. The **Evening** update shows fragments 1 and 6.

The status update can be submitted on the final page after all required fields have been filled out; the activity performs a check. A status update can also be canceled at any time. Once submitted, the episode gets stored in the local database.

3.1.5 Server-side Implementation for SlipBuddy Intervention System

The data was, at first, collected locally on the users' phones and was collected manually. Later, however, we implemented a remote database on a server to collect the data automatically to prevent us from having to have physical access to the phones that held the users' data. A server side was added for a number of reasons. The foremost was that a server would be needed to make the project more scalable. A server for syncing data allowing users from farther hospitals to join the study and have their data be valuable. Also, for the second phase, it would be necessary to have a server dispatching interventions to users dynamically using the data they were sending in. For the client side implementation, a sync service was set up using the existing Android framework for syncing data. This service sends the data to a hardcoded server in a JSON format. The sync process is triggered every time new data is entered in the client application, and will retry after a waiting period if the process fails. A sync metadata SQLite database is set up on the client, in the same configuration as the main client database, so that only new data will be sent to the server.

The sync metadata database has two tables. The location table (**Table 3.1.5.1**) contains the name of every location that was synced to the server. The syncmeta table (**Table 3.1.5.2**) is initialized with “episodes” and “question” entries, both with the “lastid” field set to 0. These rows are then updated after every successful sync operation.

Column Name	**Type**	**Description**
name	text	The name of the location

Table 3.1.5.1: Table ‘location’ Schema

Column Name	**Type**	**Description**
name	text	The name of the table
lastid	number	The ID of the last row from that table which was synced

Table 3.1.5.2: Table ‘syncmeta’ Schema

The server side implementation is an HTTPS server connected to a PostgreSQL database. The database itself is not encrypted, but it can only be directly accessed by a user logged in to the server OS. The server software is written in Java, and receives client requests and commits the data to the database. The requests are sent encrypted with the user’s password as a key. The server also has the ability to verify that a user’s password is correct by attempting to decrypt a phrase known to both the client and the server which has been encrypted using the password provided by the client. Once interventions were implemented, they were sent to the clients in the response message for a successful sync operation.

The server-side database implementation has 5 tables. The location table (**Table 3.1.5.3**) contains the real-world locations that each user has reported an episode at. A location is stored with the latitude and longitude of the location, along with a user-provided name.

Column Name	Type	Description
pkid	integer	The Primary Key
patientid	text	The ID of the user this location belongs to
name	text	Name of location
latitude	real	Latitude of the location
longitude	real	Longitude of the location

Table 3.1.5.3: Table 'location' Schema

The episodes table (**Table 3.1.5.4**) contains the overeating episodes that users report. These episodes contain all of the information asked by the application when a user reports an episode, as well as the time the episode itself was reported. This may or may not be the same as the time the episode took place, since a user can alter the time the episode took place but not the time the episode is being reported.

Column Name	Type	Description
pkid	integer	The Primary Key
patientid	text	The ID of the user this episode belongs to
id	integer	Primary key from the user's phone
timeoccur	bigint	The time the episode occurred
timestamp	bigint	The time the episode was reported

doing	text	What the user was doing when the episode occurred
doingDetail	text	Detailed text about what the user was doing when the episode occurred
eating	text	What the user was eating when the episode occurred
eatingDetail	text	Detailed text about what the user was eating when the episode occurred
stress	integer	Stress level reported for the episode
hunger	integer	Hunger level reported for the episode
location	text	The name of the location where the episode took place

Table 3.1.5.4: Table ‘episodes’ Schema

The questions table (**Table 3.1.5.5**) contains the status updates that users report. These status updates contain all of the information asked by the application when a user reports a status update, as well as which status update it was (morning, afternoon, or evening).

Column Name	**Type**	**Description**
pkid	integer	The Primary Key
patientid	text	The ID of the user this check-in belongs to
id	integer	Primary key from the user’s phone
timestamp	bigint	The time when the check-in was recorded
date	bigint	The date of the check-in
qtype	text	Which of the 3 daily check-ins this entry corresponds to
stress	integer	The stress level reported during the check-in
hunger	integer	The hunger level reported during the check-in, if applicable
sleep	integer	The sleep quality level reported during the check-in, if applicable
sleepHours	integer	The number of hours of sleep reported during the check-in, if applicable
weight	real	The weight reported during the check-in, if applicable

Table 3.1.5.5: Table ‘questions’ Schema

The interventions table (**Table 3.1.5.6**) contains the interventions that have been created for each user. Each intervention is tied to a unique user, and are represented by a formatted JSON string (see **Section 3.2.2**). Each intervention also has a flag to identify whether or not that intervention is valid.

Column Name	**Type**	**Description**
interventionid	integer	The Primary Key
patientid	text	The ID of the user this intervention belongs to
json	text	The JSON string that represents an intervention class with arguments
isvalid	boolean	The value used to determine if this intervention will be sent to the user

Table 3.1.5.6: Table ‘interventions’ Schema

The users table (**Table 3.1.5.7**) contains the ID of each user in the study, their password, and whether or not they receive interventions.

Column Name	**Type**	**Description**
id	integer	The Primary Key and the number a user signs in with
password	varchar(16)	The 16 character password a user signs in with
intervention	boolean	The value that determines if a user receives interventions

Table 3.1.5.7: Table ‘users’ Schema

3.1.6 Pilot Study

Our study was conducted by The University of Massachusetts Medical School and ran in two phases. The first phase was data collection from the users.

In order to make our application available to our test subjects, we decided to upload SlipBuddy onto the Google Play Store. The Play Store enables our application to be downloaded by our subjects without requiring the development team to use external equipment to install it, like a laptop with a wired connection. It also allows the application to update with minimal inconvenience to the user; as long as the user allows new updates from the Play Store, the app updates without losing any of the previously entered data. To limit other users from downloading SlipBuddy, we added it to the Play Store as a private beta release, which means that only people whose emails are added directly to a testing list by the development team can view and download the application. The one downfall with this method is that it requires one extra step from the user; the user needs to accept the email inviting them to be a tester in order for the application to be accessible to them. Adding the application to the Play Store was an easy task; all it required was creating the application display page, creating a signed APK, (which is the application itself), and uploading that APK to the beta testing page. Once the study began and we started receiving the subjects' emails, we added them to the testing group.

The users that signed up for our study used SlipBuddy for at least one month and were compensated for their time by The University of Massachusetts Medical School. We collected data from the first group, consisting of 4 users, and started analyzing the data in January 2016. The results of this data are discussed in **Section 3.2.1 Data Mining**. In March 2016, these 4 users will have started using SlipBuddy with interventions created specifically for them. After these 4 users finished the first part of the study, The University of Massachusetts Medical School started recruiting more users online, which would allow for more data mining of individual users' data and for a more definitive look at the general population.

3.2 Phase 2

Phase one was mainly designed to test the usability of the SlipBuddy app, and to collect data on several users to begin development for the second phase. The key feature of the second phase is the intervention. The idea for an intervention is that if we can know a user will overeat before they do it, and then intervene and warn the user, then the user will be better equipped to make a healthier choice. That leaves us with three key goals:

1. Use data mining to predict overeating episodes,
2. Design effective interventions, and
3. Implement a way of delivering interventions to the users.

3.2.1 Predicting Overeating Episodes through Data Mining

Standard data mining techniques were used to try to predict whether or not SlipBuddy users were likely to overeat based on how they responded to survey questions posed in the SlipBuddy application. Algorithms such as Decision Trees and Logistic Regression were used to see what factors were most influential and which factors were the best predictors. We used the implementation of Decision Trees (called J48) and of Logistic Regression available in the Weka system version 3.6 [8] [9]. Our team also went over the data by hand to see what correlations we could find that might be heuristically useful for predicting overeating episodes.

General Data Exploration

In addition to algorithms, our team tried to look by hand at the data and find patterns that might not come up as significant in a more statistical approach. This method was also quite fruitful, as we found many patterns. For reference, the red bars in the graphs show overeating days, and the blue bars shows days without overeating.

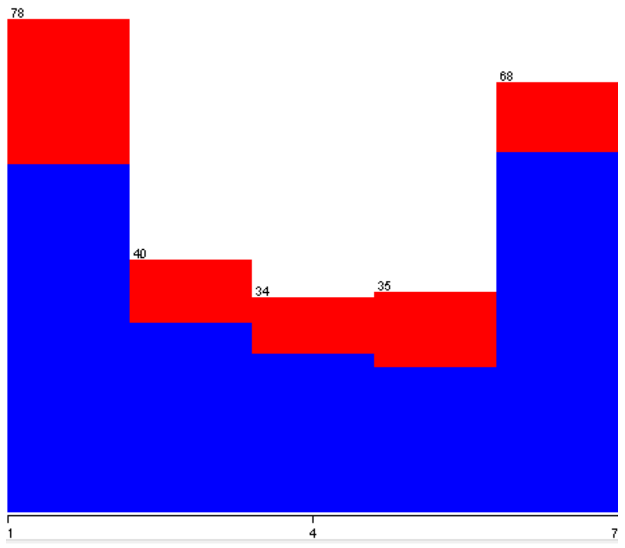


Figure 3.2.1.1: Days of the Week

Figure 3.2.1.1 starts with Monday on the left and ends at Sunday on the right. From the figure, it is clear that people are far more likely to over eat at the start of the work week.

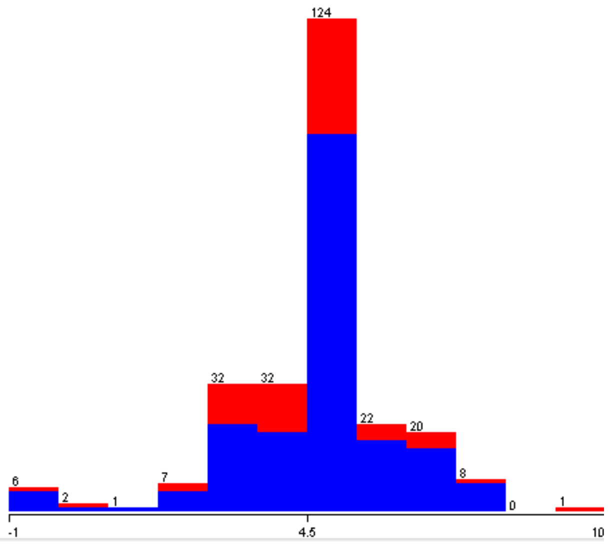


Figure 3.2.1.2: Hunger

From **Figure 3.2.1.2**, we can see that hunger levels 4 and 5 are valuable for predicting overeating episodes. 5 is the default hunger value in the app, and also a good middle value for when someone is not notably full or hungry. This feature explains why 5 is far more common than any other entry.

Decision Trees Interpretation

The Decision Tree algorithm is used to predict outcomes based on a number of factors by generating a tree and attempting to isolate positive and negative cases based on combining the factors, and measuring how well the tree was able to predict outcomes. These trees were generated using the decision tree mining algorithm implemented in the Weka data mining tool, called J48. For the purpose of this paper we will only explore the J48 tree generated with all of the patients data, as shown in **Figure 3.2.1.3**.

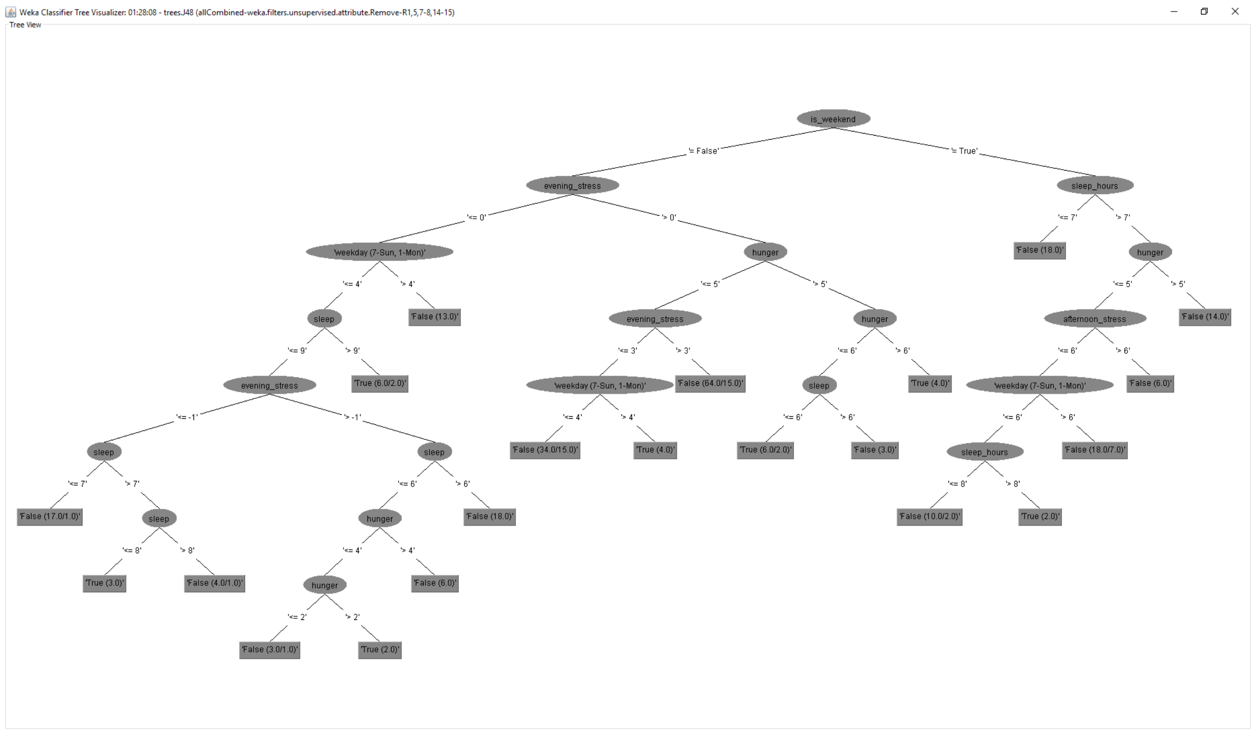


Figure 3.2.1.3: Full J48 Decision Tree

Let us break down the most important parts of the tree. The top are generally the most influential features, since their ability to predict outcomes is greater.

As **Figure 3.2.1.4** shows, whether or not it was the weekend is the most influential. J48 found that it is very uncommon for people to overeat on the weekend, and knowing this pattern it could easily predict many days as not having overeating episodes based almost completely on that. Due to the uncommon nature of a weekend overeating episode, we will only explore the tree created for weekdays.

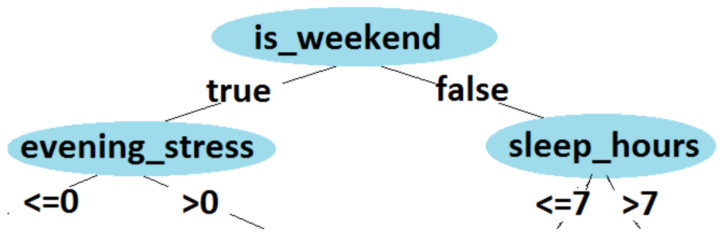


Figure 3.2.1.4: Top-Level of the J48 Decision Tree

The next strongest factor is evening_stress, as shown in **Figure 3.2.1.5**. 0 was the most common entry for users, which is most likely due to them not feeling any notable stress at the time of taking the survey. In the case where the user is relaxed, the tree turns to the day of the week, and there is a clear bias toward the start of the week, as any day during the week after Thursday is likely to be a day without overeating.

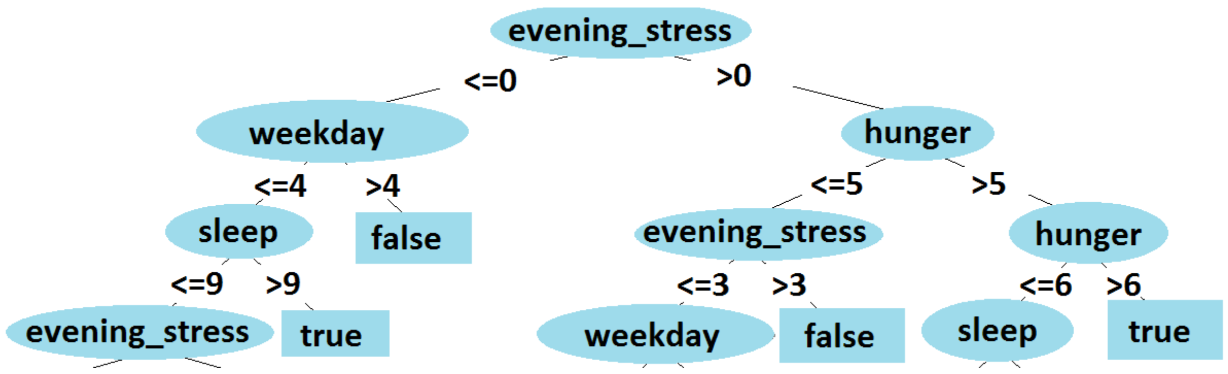
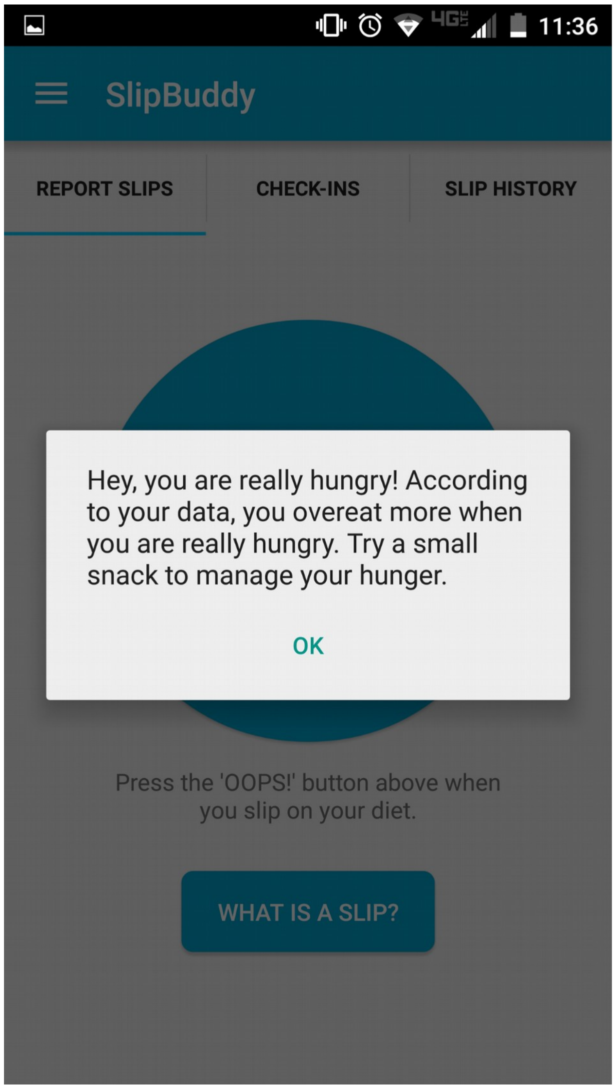


Figure 3.2.1.5: Mid-Level branch of the J48 Decision Tree

On the other hand, when users found their stress level to be notable (above 0), the next most valuable factor becomes a person’s hunger. Here the combination of hunger and stress lead to deciphering days when overeating episodes occurred.

3.2.2 Designing Interventions

With data on when overeating episodes happen and what influenced them, we set eyes on the goal of warning users when they are in danger of overeating. The hypothesis here is that if we are able to predict overeating episodes, and warn users that they are in danger of having an overeating episode, then people will become more aware of their dangerous eating habits. This should make overeating episodes more infrequent, and in turn will make people begin to lose weight. With the data and predictions in place we looked to designing a quick and simple interface that isn’t intrusive, but still quickly and cleanly communicates the user’s situation. See an example in **Figure 3.2.2.1**.



Hey, you are really hungry! According to your data, you overeat more when you are really hungry. Try a small snack to manage your hunger.

Press the 'OOPS!' button above when you slip on your diet.

Figure 3.2.2.1: Intervention Dialog Box

Intervention Triggers

In order to keep the Interventions as modular as possible, we needed to abstract the implementation of the trigger logic away from the core Intervention code. To this end, we created an Interface called InterventionTrigger that trigger classes could then implement. The main feature of this interface is the shouldTrigger method, which tells the Intervention whether or not it is an appropriate time to trigger, with the implementing logic left up to the trigger class. There is also a triggerLess method, which was originally intended to tell the InterventionTrigger to trigger less often. Though we decided to remove this feature, as it didn’t make sense for any of our current Interventions, the code paths to support it still exist. After either of these methods is called, the main Intervention object makes sure to update the serialized copy of the object in the database, so as to maintain the trigger’s internal state.

Initially, we designed trigger classes that were tailored to each user. However it was quickly determined to be unsustainable. To address this, we designed the GenericTrigger class, which allows for a more direct mapping from our generated decision trees to Interventions. The GenericTrigger allows for testing if certain data fields are in a specified range, if they equal a certain value, or if certain conditions are met. GenericTriggers can also be nested, in order to handle more complex conditions. Generally speaking, each GenericTrigger recursively implements a single path in a decision tree. The currently implemented conditions are detailed in **Table 3.2.2.1**.

Field Name	What it Checks
HUNGER	Checks the hunger value reported in the morning check-in. Expects an int. Supports both a range or an expected value.
EPISODE_TODAY	Checks if an episode has occurred today. Expects a boolean. Only supports a single expected value.
AFTERNOON_STRESS	Checks the stress level reported in the afternoon check-in. Expects an int. Supports both a range or an expected value.
SLEEP	Checks the value for sleep quality reported during the morning check-in. Expects an int. Supports both a range or an expected value.
WEEKDAY	Checks which day of the week it is. Expects an int. 1 is Monday, and 7 is Sunday. Only supports a single expected value.
MORNING_STRESS	Checks whether or not it is the weekend. Expects a boolean. Only supports a single expected value
EVENING_STRESS	Checks the stress level reported in the evening check-in. Expects an int. Supports both a range or an expected value.
SLEEP_HOURS	Checks the number of hours slept reported in the morning check-in. Expects an int. Supports both a range or an expected value.
HOUR	Checks the current hour. Expects an int. Hours are base on a 24-hour clock. Supports both a range or an expected value.
IS_WEEKEND	Checks whether or not it is the weekend. Expects a Boolean. Only supports a single expected value.

Table 3.2.2.1: Generic Trigger Fields

Intervention Service

For the implementation of interventions on the client, all of the functionality was segmented off into an Android BroadcastReceiver called InterventionService that runs in its own process. An Intervention class was created to represent the interventions in memory. This class contains the intervention’s ID (which is unique for users), the text that should be displayed when the Intervention is triggered, and a reference to an object that implement our InterventionTrigger Interface. The type of InterventionTrigger which is used decides the exact logic for when the intervention should be triggered. A list of current interventions is kept on the phone in a SQLite database, which is set up the same way as the main client database, stored as the ID of the intervention and a serialized form of the object in JSON, generated using the GSON library.

There are two cases where the InterventionService receives a broadcast. The first is when new data is received from the server by the sync service. The sync service sends the InterventionService a list of Interventions in a high level JSON representation, which is converted into a List of Intervention objects via the InterventionFactory class. When an Intervention that is currently in the local list is in the new list, no action is taken. When one is only present in the new list, it is added to the local list. When one is present on the local list, but not the list from the server, it is considered expired, and is removed from the local list. The second case is when a new episode or status update is recorded, so that updated data from those can be used by the InterventionTriggers to decide if one should trigger. The InterventionService can only serve one broadcast at a time, so new broadcasts coming in are served in a first in, first out manner.

The schema used for the client side Intervention database is represented in **Table 3.2.2.2**. This table holds the ID of the intervention and the serialized version of the Intervention data.

Column Name	**Type**	**Description**
id	number	Primary key
data	text	Serialized Intervention instance

Table 3.2.2.2: Table “interventions” schema

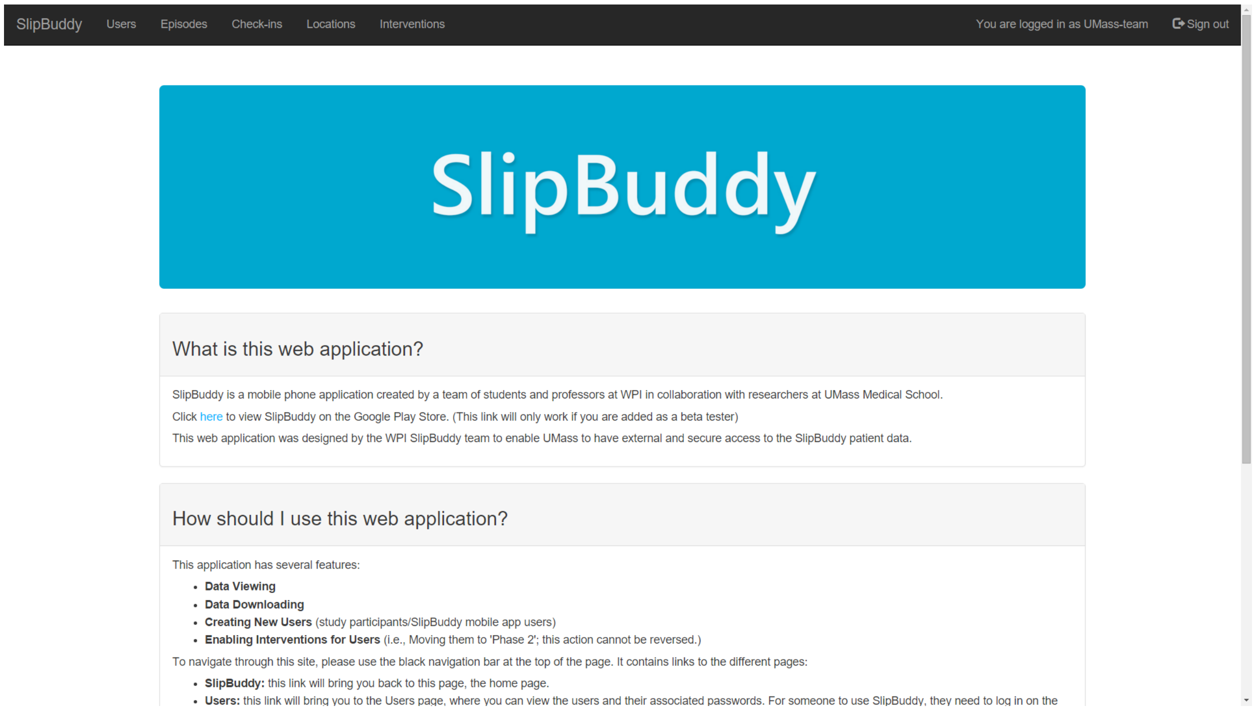
3.2.3 Implementing Interventions

In order to display Interventions the client must first receive a list of Interventions from the server. To this end, the previously unused response body during synchronization is utilized. The response body contents are now a JSON formatted array of Interventions. It should be noted that this format is not directly a serialized form of the object, but instead an array of JSON objects which can be used to construct the Intervention object on the client side.

In order to tell if a user should receive Interventions, a Boolean column was added to the users table in the server database. If the value is true, they get interventions. If it's false, they get served an empty array. Each user's Interventions are stored on the server in an interventions table. This table has 4 columns, the Intervention's ID, the user ID it corresponds to, a flag so that the Intervention can be disabled, and the JSON representation to be sent to the client.

3.3 SlipBuddy Provisioning Application

Once we started receiving data from the SlipBuddy App, we realized that we needed an efficient way to view and control this data. We also needed to allow The University of Massachusetts Medical School researchers to view the data remotely and securely. In order to meet these goals, we implemented a web application (**Figure 3.3.1**) that would allow authenticated users the ability to view all of the data stored on our server.



What is this web application?

SlipBuddy is a mobile phone application created by a team of students and professors at WPI in collaboration with researchers at UMass Medical School.

Click here to view SlipBuddy on the Google Play Store. (This link will only work if you are added as a beta tester)

This web application was designed by the WPI SlipBuddy team to enable UMass to have external and secure access to the SlipBuddy patient data.

How should I use this web application?

This application has several features:

- **Data Viewing**
- **Data Downloading**
- **Creating New Users** (study participants/SlipBuddy mobile app users)
- **Enabling Interventions for Users** (i.e., Moving them to 'Phase 2'; this action cannot be reversed.)

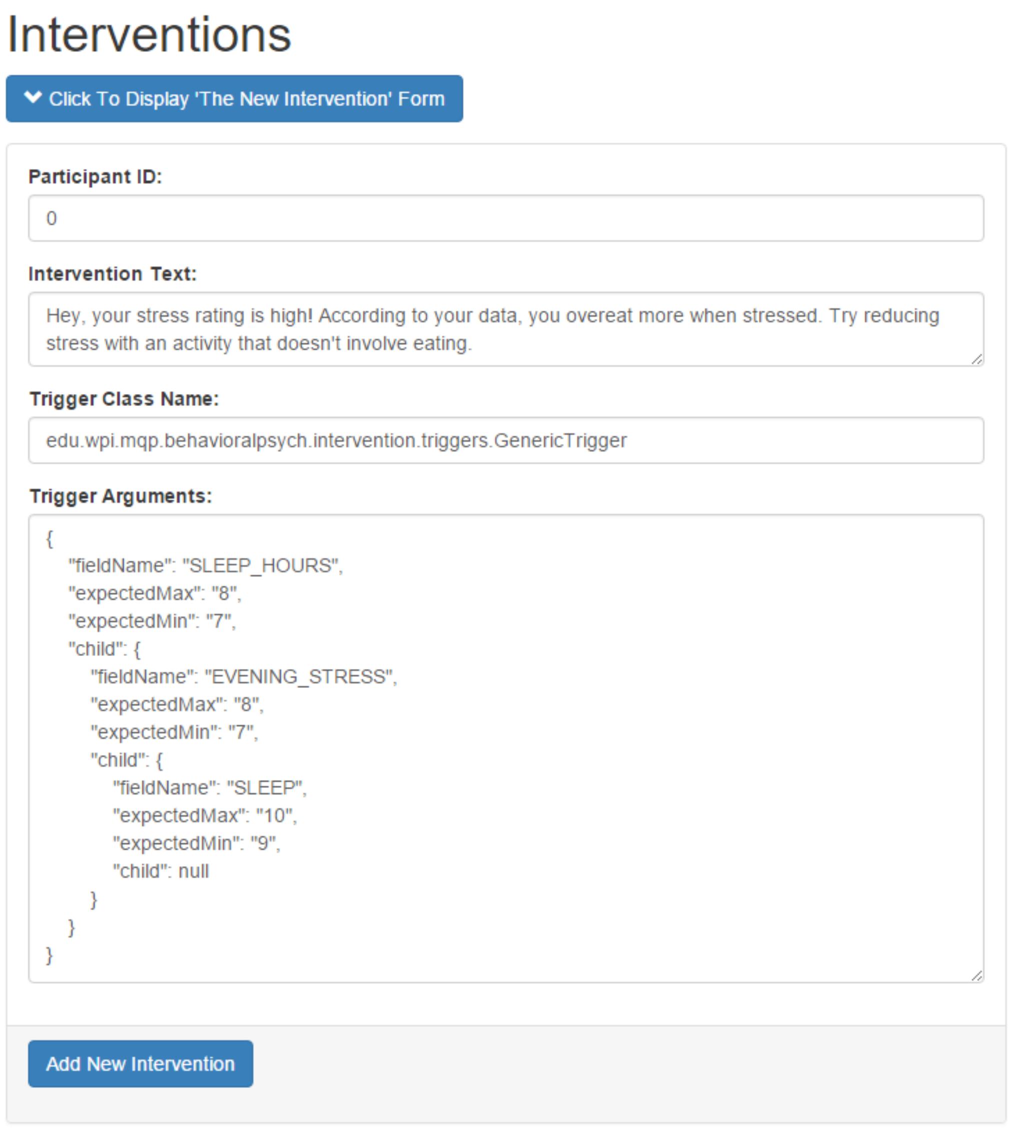
To navigate through this site, please use the black navigation bar at the top of the page. It contains links to the different pages:

- **SlipBuddy:** this link will bring you back to this page, the home page.
- **Users:** this link will bring you to the Users page, where you can view the users and their associated passwords. For someone to use SlipBuddy, they need to log in on the

Figure 3.3.1: The SlipBuddy Provisioning Application

The web application running on the same machine as the server for the SlipBuddy app, was created using Node.js. and Jade templates. It connects to our server's PostgreSQL database in order to retrieve and interact with the data. To protect the privacy of the user's data, we have implemented several security features, such as using HTTPS with certified credentials and users with secure passwords. Only two users were created: one for the WPI team and one for the University of Massachusetts team. The passwords were exchanged using secure methods.

The web application, sometimes referred to as the SlipBuddy Provisioning Site, offers the ability to view SlipBuddy user IDs and their associated passwords, overeating episodes, check-ins, locations, and the created interventions. New users can be created with auto-generated passwords and there is a button to enable a SlipBuddy user to start receiving interventions. There is also the ability to create new interventions on the 'Interventions' page, as shown in **Figure 3.3.2**, but this option is only available to the WPI team, since it requires technical knowledge about our code to use. Another feature only available to the WPI team is the ability to look at the SlipBuddy Javadocs, which contains information on how to create these interventions.



Interventions

Click To Display 'The New Intervention' Form

Participant ID: 0

Intervention Text: Hey, your stress rating is high! According to your data, you overeat more when stressed. Try reducing stress with an activity that doesn't involve eating.

Trigger Class Name: edu.wpi.mqp.behavioralpsych.intervention.triggers.GenericTrigger

Trigger Arguments:

```
{
    "fieldName": "SLEEP_HOURS",
    "expectedMax": "8",
    "expectedMin": "7",
    "child": {
        "fieldName": "EVENING_STRESS",
        "expectedMax": "8",
        "expectedMin": "7",
        "child": {
            "fieldName": "SLEEP",
            "expectedMax": "10",
            "expectedMin": "9",
            "child": null
        }
    }
}
```

Add New Intervention

Figure 3.3.2: The New Intervention Form

4 Future Work

The time period for this MQP, consisting of only 3 quarters, has limited how far this project can be developed and analyzed. This section describes the work that could be done in the future. In a nutshell, more data could be collected and analyzed, processes could be made automatic, and new features could be added.

4.1 Further Data Analysis

As this project concludes, data is still coming in from the SlipBuddy pilot study as new users are added to the project. The data analysis for phase 2 was done using only 4 users. Fortunately, we designed interventions individually for each user, so this was not much of a hindrance. However, for this project to grow, future analysis needs to be done with at least 30 users to ensure statistical significance.

4.2 Automatic Intervention Management

Another aspect of SlipBuddy that needs to be improved upon is that, at the moment, interventions are designed and added to the database by hand. We decide what is valuable for the intervention, as well as how it should be implemented for the user. In the future, this process should be automated, possibly using Weka's libraries to streamline the process. In order for this program to be truly scalable and effective, the process of creating useful interventions needs to happen without human input. This could be done entirely server side, and the phone application would not need to be changed at all. The process would only need to dynamically add interventions to the database.

4.3 Additional Features

SlipBuddy currently is recording different types of data, such as stress, hunger, and sleep, and those values are being used to create interventions for each individual user, but there is other data that is recorded but not considered, such as location data. Location data is recorded when an overeating episode is reported, but we did not use this data during our intervention creation process. This could be implemented by having the intervention service running more frequently by polling the phone's location, if the user hasn't disabled location data use in the settings menu. A new generic trigger could be made that accounts for location proximity that the intervention service would check. Adding this feature would raise a few questions. How often should the service check to remain responsive, but without draining a phone's battery? How close would the app want someone to be to the location before it triggers? How would the app handle faulty location data? And, would the app ignore specific locations, such as 'Home'? These questions would have to be answered before this feature could be implemented.

SlipBuddy could also add features to visually represent a user's progress over time. For example, the app could display a user's weight, number of overeating episodes, or stress over time. It could provide graphs that are specific to their interventions. So, if a user often overeats when they have high stress, there could be a graph showing their overeating episodes on a stress graph. Participants using the app could be asked if they would find this feature useful.

4.4 Summary

SlipBuddy currently is doing a massive amount of data collection, but there is definitely a lot more that can be done to make this a more refined product. The data coming in needs to be analyzed, the analyzing process needs to be made automatic in order for the app to be scalable, and new features can be added that handle different types of interventions as well as provide more feedback for the user.

To conclude, SlipBuddy is a good first step into exploring how mobile technology can be used to encourage weight loss. Future developers should flesh out the app’s features and the results of using the app by patients. We hope that, in the future, SlipBuddy can be used as a tool to help those who struggle with overeating.

5 Works Cited

[1] "Overweight and Obesity Statistics." [Online]. Available: http://www.niddk.nih.gov/health-information/health-statistics/Pages/overweight-obesity-statistics.aspx. [Accessed: 22-Mar-2016].

[2] "Bringing more effective tools to the weight-loss table," *http://www.apa.org*. [Online]. Available: http://www.apa.org/monitor/jan04/bringing.aspx. [Accessed: 22-Mar-2016].

[3] *Google Opinion Rewards*. Google Inc.

[4] S. Pagoto, K. Schneider, M. Jojic, M. DeBiasse, and D. Mann, "Evidence-Based Strategies in Weight-Loss Mobile Apps," *American Journal of Preventive Medicine*, vol. 45, no. 5, pp. 576–582, Nov. 2013.

[5] C. A. Pellegrini, J. M. Duncan, A. C. Moller, J. Buscemi, A. Sularz, A. DeMott, A. Pictor, S. Pagoto, J. Siddique, and B. Spring, "A smartphone-supported weight loss program: design of the ENGAGED randomized controlled trial," *BMC Public Health*, vol. 12, p. 1041, 2012.

[6] S. Michie, S. Ashford, F. F. Sniehotta, S. U. Dombrowski, A. Bishop, and D. P. French, "A refined taxonomy of behaviour change techniques to help people change their physical activity and healthy eating behaviours: The CALO-RE taxonomy," *Psychology & Health*, vol. 26, no. 11, pp. 1479–1498, Nov. 2011.

[7] H. Oinas-Kukkonen and M. Harjumaa, "Persuasive systems design: Key issues, process model, and system features," *Communications of the Association for Information Systems*, vol. 24, no. 1, p. 28, 2009.

[8] I. H. Witten, E. Frank, and M. A. Hall, *Data mining: practical machine learning tools and techniques*, 3rd ed. Burlington, MA: Morgan Kaufmann, 2011.

[9] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The WEKA data mining software: an update," *ACM SIGKDD explorations newsletter*, vol. 11, no. 1, pp. 10–18, 2009.