Software Design Document

for

Urban Earth iOS Application, Version 1.0

Prepared by:
Jeffrey Kozik
Niki Wang

October 5, 2021

Table of Contents

1. INTRODUCTION

1.1 Content of the Document

In this document, the below aspects of the Urban Earth iOS Application software architecture are described:

- Principal Classes
- Interactions between Components (Inter-agent Protocols)
- Class Interfaces
- Inspection Log

So that the code is easy to maintain and easy to change as features change, we use Object Oriented programming. Additionally, we use Swift as our programming language and Xcode for the GUI as these are the tools native to App Development on iOS.

1.2 Vision & Scope

Due to the ecological impact done by industrialization, and specifically air pollution caused by internal combustion engine, the Urban Earth iOS Application dedicates the goal of making a positive impact around the community, serves to incentivize people to travel in a sustainable way, and promotes the importance of contributing one’s part to preserve the elegance and beauty of the Mother Earth.
According to the Software Requirements Specifications for Urban Earth iOS Application release 1.0, the Urban Earth iOS Application is used to track how often the user travels sustainably and shows the amount of CO_2 prevented from emitting into the atmosphere per trip, the number of trees planted, or the number of polar bears saved per user choice. These units of choice can be counted towards the user's sustainability scores. Additionally, the application allows for users to be “friends” with other users to see the “sustainability score” of their friends and build an encouraging community. Because one of the main forms of sustainable transportation, walking, can be potentially more dangerous at night than non-sustainable transportation options such as driving, the app also issues alerts to users when to be careful and attentive to their surroundings. Such intended major features shall be delivered to users over multiple releases.

2. APPLICABLE DOCUMENTS

Software Requirements Specifications for Urban Earth iOS Application, Release 1.0, Prepared by: Jeffrey Kozik, Niki Wang, September 21, 2021.

The official U.S. Government Source for Fuel Energy Information, the U.S. Department of Energy, the U.S. Environmental Protection Agency, www.fueleconomy.gov, updated on October 5th, 2021.

Frequently Asked Questions, U.S. Energy Information Administration, www.patagoniaalliance.org/wp-content/uploads/2014/08/How-much-carbon-dioxide-is-produced-by-burning-gasoline-and-diesel-fuel-FAQ-U.S.-Energy-Information-Administration-EIA.pdf, updated on May 21st, 2014.

Fuel Efficiency: Modes of Transportation Ranked By MPG, True Cost, praveenghanta, truecostblog.com/2010/05/27/fuel-efficiency-modes-of-transportation-ranked-by-mpg/, updated on May 27, 2010.

Fueleconomy, Github, ryankirkman, github.com/ryankirkman/fueleconomy, updated on November 8th, 2015.

3. PRINCIPAL CLASSES

3.1 Responsibilities

The requirements for the Urban Earth iOS Application include the following responsibilities:

R1. Recording the mileage of a user’s trip
R2. Recording the form of sustainable transportation for each trip
R3. Calculating the sustainability score per trip to display it after a trip ends
R4. Storing the sustainability score and summing the sustainability scores over time
R5. Storing the history of time duration, mileage, form of transportations, sustainability scores for a user’s trips
R6. Handling changes, addition, and edition of a user’s trips
R7. Displaying graphs of a user’s sustainability score over time
R8. Alerting users to stay attentive and aware of their surroundings at night
R9. Processing a user’s requests to add, search, or delete a friend among one’s community
R10. Displaying the sustainability score of a user’s friend per request

According to the Software Requirements Specifications for Urban Earth iOS Application release 1.0, some of the major features FE1-7 but not all define the responsibilities of the Urban Earth iOS Application. The UE iOS Application records the mileage of a user's trip and form of transportation based on FE-1 and FE-3 in the *User* object. The iOS Application calculates the sustainability score per trip based on FE-2 in the *User* object. Responsibility R6 allows users to make changes for their trips based on FE-4 in the *User* object. Responsibility R7 allows users to graph their sustainability score over time based on FE-5 in the *Graph* object. Responsibility R8 alerts users to be attentive at night based on FE-6. Responsibility R9 and R10 allows users to build community within the Urban Earth application based on FE-7 in the *User* object. Responsibility R11 allows data to be stored in a secure way in the *User* object.

3.2 User

In the current design, there is a User class, which is primarily responsible for the following:

R1. Storing and being able to modify the user's email address
R2. Storing and being able to modify the user's display name
R3. Storing and being able to add or remove the user's friends
R4. Storing and being able to modify the user's total sustainability score
R5. Storing and being able to modify all of the user's trips
R7. Storing and being able to modify if the user wants trips to be auto tracked or only manually recorded
R6. Storing and being able to modify the user's default form of transportation
R7. Storing and being able to modify the user's "quick pick" forms of transportation
R7. If the user allows auto-tracking of the trip, being able to detect when the trip begins
R8. If the user allows auto-tracking of the trip, being able to detect when the trip ends
R9. Storing and being able to modify what the default sustainability score unit is for the user (for example: pounds of CO2, kg of CO2, tons of CO2, # of polar bears saved, # of trees planted)

Responsibility R4 updates the total sustainability score by getting the sustainability score from the latest trip and adding it to the total sustainability score. If the fields in a specific trip are modified, this will result in a modified sustainability score for that trip which results in a modified total sustainability score for the user. R5 stores objects of type Trip. R6 and R7 store objects of type Transportation. R9 uses the name of the Unit from Unit.

3.3 Trip

There is a Trip class corresponding to each time a user travels:

R1. Storing whether this trip has happened in the past or is currently ongoing

R2. Storing when the trip started and ended using both time and date
R3. If this trip is ongoing, live updating the distance traveled in this trip
R4. Storing and being able to manually modify the trip's distance after the fact
R5. Storing the type of transportation being used (for example: walking, biking/scootering, electric biking/scootering/segway, electric car, hybrid car, bus, subway, plane) and being able to manually modify the type of transportation after the fact
R6. Storing the sustainability score of the trip and being able to modify the sustainability score (not directly by the user, but if the form of transportation and/or the distance traveled is changed, the sustainability score should auto-update.

Responsibility R5 stores objects of the type Transportation. R6 uses the factor in the unit to calculate the score.

3.4 Transportation

R1. Storing and being able to modify the transportation's name

3.4.1 Car

R1. Storing and being able to modify what the car's make and model is
R2. Storing and being able to modify what the car's mpg is

Car is a subclass Transportation.

3.4.2 Non Car

R1. Storing and being able to modify what type of transportation it is (walking, biking, e-scooter, public transport, etc)
R2. Storing and being able to update what the non-car's mpg is

Non-car is a subclass Transportation.

3.5 Graph

R1. Storing and being able to modify the time range (from start date to end date)
R2. Storing and being able to modify the interval (days, weeks, months, years)
R3. Storing and being able to modify the unit
R3 uses the name of Unit from Unit to display on the Graph.

3.6 Unit

R1. Storing and being able to modify the name of the unit
R2. Storing and being able to modify the factor of the unit

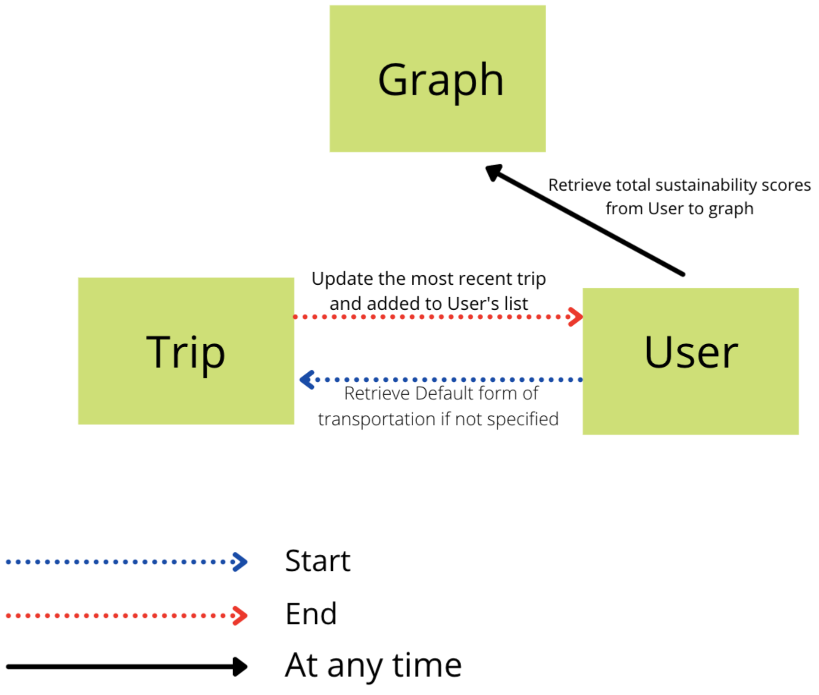
Graph
Retrieve total sustainability scores from User to graph
Trip
Update the most recent trip and added to User's list
User
Retrieve Default form of transportation if not specified
Start
End
At any time

Figure 1: Demonstration of interactions among *Trip*, *User*, and *Graph*

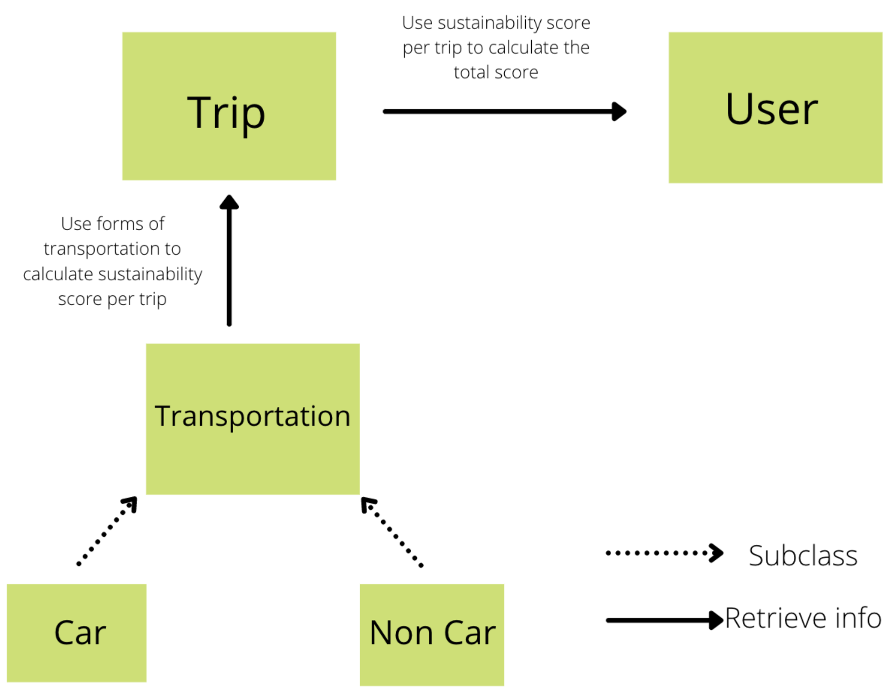
Trip
Use sustainability score per trip to calculate the total score
User
Use forms of transportation to calculate sustainability score per trip
Transportation
Car
Non Car
Subclass
Retrieve info

Figure 2: Demonstration of interactions among *Car*, *Non Car*, *Transportation*, *Trip*, and *User*

4. INTER-AGENT PROTOCOLS

4.1 Starting a trip

When a *Trip* is started (either manually or automatically), if a type of sustainable transportation isn't specified, the *User*'s default form of transportation is used.

4.2 Ending a trip

When a *Trip* is ended (either manually or automatically), the trip has to be added to the *User*'s list of trips. Furthermore, the *User*'s total sustainability score has to be updated by first calculating the *Trip*'s sustainability score, then adding that score to the *User*'s current total sustainability score.

4.3 Modifying the form of transportation being used in a trip

When the type of transportation being used in a *Trip* is modified, the old sustainability score of the *Trip* must be first subtracted from the *User*'s total sustainability score, then the *Trip*'s sustainability score must be recalculated and finally the new sustainability score is added to the *User*'s sustainability score.

4.4 Modifying the distance traveled in a Trip

When the distance traveled of a *Trip* is modified, the old sustainability score of the *Trip* must be first subtracted from the *User*'s total sustainability score, then the *Trip*'s sustainability score must be recalculated and finally the new sustainability score is added to the *User*'s sustainability score.

4.5 Graphing sustainability score over time

When a new *Graph* is created, it gets all of the *Trip*s in the relevant time period from the *User*'s trips field. By default, the graph will get the *User*'s default sustainability unit to graph by.

4.6 Calculating Sustainability Score

When a sustainability score is being calculated for a *Trip*, it is calculated by using the miles per gallon field from the Transportation class that was in the *Trip*.

5. CLASS INTERFACES

5.1 User

5.1.1 Public Method GetEmail

String GetEmail()

The GetEmail() method returns a User's email

5.1.2 Public Method SetEmail

Void SetEmail(String email)

The SetEmail() method modifies a User's email

5.1.3 Public Method GetDisplayName

String GetDisplayName()

The GetDisplayName() method returns a User's display name

5.1.4 Public Method SetDisplayName

Void SetDisplayName(String displayName)

The SetDisplayName() method modifies a User's display name

5.1.5 Public Method addFriend

Void addFriend(User friend)

The addFriend() method adds a User to the current User's list of friends by their email (no to user's can sign up for an account using the same email)

5.1.6 Public Method removeFriend

Void removeFriend(User friend)

The removeFriend() method removes a User from the current User's list of friends by their email

5.1.7 Public Method getSustainabilityScore

int getSustainabilityScore()

The getSustainabilityScore() method gets the current user's total sustainability score

5.1.8 Public Method setSustainabilityScore

void setSustainabilityScore()

The setSustainabilityScore() method modifies the current user's total sustainability score (this can't be manually done by the user, but happens when the User's list of trips changes in some way)

5.1.9 Public Method addTrip

void addTrip(Trip trip)

The addTrip() method adds a trip to the current user's list of trips

5.1.10 Public Method removeTrip

void removeTrip(Trip trip)

The removeTrip() method removes a trip from the current user's list of trips

5.1.11 Public Method getAllowsAutoTrack

boolean getAllowsAutoTrack()

The getAllowsAutoTrack() method returns whether or not the user allows trips to be autotracked or just allows them to be manually tracked

5.1.12 Public Method setAllowsAutoTrack

void setAllowsAutoTrack()

The setAllowsAutoTrack() method modifies whether or not the user allows trips to be autotracked

5.1.13 Public Method getDefaultTransportation

Transportation getDefaultTransporation()

The getDefaultTransportation() method gets what the user's default mode of transportation is

5.1.14 Public Method setDefaultTransportation

void setDefaultTransportation(Transportation transportation)

The setDefaultTransportation() method modifies which transportation mode is the user's default

5.1.15 Public Method addQuickPickTransportation

void addQuickPickTransportation(Transportation transportation)

The addQuickPickTransportation() method adds a form of transportation to the quick picks the user can easily select from

5.1.16 Public Method removeQuickPickTransportation

void removeQuickPickTransportation(Transportation transportation)

The removeQuickPickTransportation() method removes a form of transportation from the quick picks the user can easily select from

5.1.17 Public Method detectTripStart

void detectTripStart()

Detects when trip begins, when it does creates a new Trip object (only runs when autoDetect is enabled)

5.1.18 Public Method detectTripEnd

void detectTripEnd()

Detects when trip ends, when it does changes the field in Trip to indicate the trip is no longer active

5.1.19 Public Method getUnit

String getUnit()

Returns a String representing the default unit to be used for sustainability scores

5.1.20 Public Method setUnit

void setUnit(String unit)

Sets the default Unit for the user

5.1.21 Public Method sendAlert

void sendAlert(Date time)

Sends an alert to the users based on the time

5.2 Trip

Not all Getters and setters not listed (refer to Unit class to see how these type of methods will work)

5.2.1 Public Method calculateSustainabilityScore

int calculateSustainabilityScore()

Calculates the sustainability score for the Trip based upon the distance traveled and the vehicle used as well as the location of the user. Refer to Applicable Documents to read more about how this process will work.

5.2.1 Public Method getTime

Date getTime()

The getTime() method gets the time of the trip and returns a Date

5.2.2 Public Method setTime

void setTime()

The setTime() method modifies the date and the time of the trip

5.3 Transportation

5.3.1 Public Method getTransportation

string getTransportation()

This method returns the transportation name as a string

5.3.2 Public Method setTransportation

void setTransportation(string transportation)

This method modify the transportation name

5.4 Car

5.4.1 Public Method getModel

string getModel()

This method returns the model of the car as a string

5.4.2 Public Method setModel

void setModel(string model)

This method takes a string and modifies the model of the car

5.4.3 Public Method getMpg

Double getMpg()

This method returns the mpg of the car

5.4.4 Public Method setMpg(Double mpg)

This method takes a mpg and modifies the mpg

5.5 Non Car

5.5.1 Public Method getNoncarmodel

string getNoncarmodel()

This method returns the model of the non-car as a string

5.5.2 Public Method setNoncarmodel

void setNoncarmodel(string model)

This method takes a string and modifies the model of the non-car

5.5.3 Public Method getNoncarmpg

Double getNoncarmpg()

This method returns the mpg of the car

5.5.4 Public Method setNoncarmpg(Double mpg)

This method takes a mpg and modifies the mpg

5.6 Graph

Methods not listed as they are only getters and setters (refer to above classes for how these will work)

5.7 Unit

Methods not listed as they are only getters and setters (refer to above classes for how these will work)

6. INSPECTION LOG

Name	Comment
Niki Wang	10/5/2021 - the list of all getter and setter methods in the principal classes is a bit redundant and could be clarified with one sentence.

Jeffrey Kozik	10/5/2021 - could modify the “responsibilities” slightly so that the same responsibilities are listed in the responsibilities header and for each of the Classes, however the way it’s done does make sense so that the classes can get more in depth of exactly what they’re doing Could remove getters and setters in all the class interfaces as these methods are commonly understood