1. DESCRIPTION OF THE SYSTEM

CO2 Buddy is a cloud-based system using three external systems and it is accessed through a mobile application and a single page application. The system's internal components use AWS's on-demand services. Such services are used in user login, the API and database.

The system allows the user to connect their All Receipts™ account from which the system will get data about the users' receipts. The data from the receipts will be stored in the database and after this the CO2 footprint will be calculated according to how much CO2 emissions each item on the receipt has produced. Our system is also able to calculate the users CO2 footprint from their travelling habits. It also gives the user feedback according to their purchases and ways of travelling. The system also makes it possible to compensate for the CO2 footprint. It stores compensations options in the database that are then displayed for the user. User can then select the amount of compensation and after which the payment will be handled through PayPal. The compensated CO2 amount will then be stored in the users' own information.

2. DESCRIPTION OF CHOSEN ARCHITECTURE MODEL

The C4 model visualises the architecture of a system on multiple abstraction levels. The name comes from the four levels: context, containers, components and code. The levels portray the system on different levels of detail and are aimed for different audiences.

On context level the system is visualised as one box and the important part is the external connections with your system. Containers level zooms in the system and displays the containers (separately runnable/deployable units that execute code or store data) and their connections. On components level each container is divided to components and their responsibilities and technology details are presented. On the last level, code, an implementation as code is visualised for each component. [1]

We were given two models as a suggestion. We chose to pick a suggested model for learning purposes and thinking that maybe one of them could be in the exam of the course. Also, someone more experienced than us made those suggestions.

Particularly C4 we chose based on a quick look on both models. The C4 model seemed simpler and more understandable. The web page for C4 had clear example pictures of the model and we thought it was the right choice for us.

3. DESCRIPTION OF AWS

3.1. GENERAL

Amazon Web Services (AWS) is a cloud-computing platform with many on-demand services they provide. The services are serverless to the user, they scale automatically to

the usage and are billed according the use. AWS serves a full array of functionalities without the need to own huge server centres. [2, 3]

We chose to use AWS in this project, because of the requirements of uptime, security and strong authentication. AWS was somewhat familiar to all our group members and it's the market leader on cloud platforms.

3.2. COGNITO

AWS Cognito is a simple user authentication service from AWS. It provides a user directory, third party sign in option (Facebook, Google…) and easy integration to applications. [4]

We have decided to use Cognito with JavaScript framework Amplify. Amplify is familiar to one of the group members. Amplify simplifies the use of Cognito in code. Cognito can also be used with the AWS web client, but to fully utilise the Cognito features using some tool is handy. [5]

3.3. DATABASE

AWS offers multiple solutions for a database. You can choose one for your needs. Databases scale automatically and are billed per usage. Databases are also fully managed. [6]

Since we have decided to use AWS, it was an easy decision to use AWS databases. Scalability, low maintenance and multiple options were also pros.

3.4. API GATEWAY

AWS API Gateway is the service to create and maintain API's in AWS. API Gateway has also ways to monitor API traffic and visualises it for the developer. API Gateway has two types to choose from: RESTful and Websocket API. The API Gateway can also be used with the Cognito to restrict user's access to resources. [7]

We have chosen to use a RESTful API Gateway API with our Cognito user control. API Gateway is also familiar to one of our group members, which did affect out decision.

4. DESCRIPTION OF EXTERNAL SYSTEMS

4.1. GOOGLE MAPS PLATFORM

Google Maps Platform is a dynamic map and route services. In our system we use Google Maps Routes to get directions and travel durations. Directions and durations are queried from a developer API.

Directions API is queried with the traveller's route's origin and destination and the transportation type, which is in our case public transportation. [10] Duration matrix API is queried with the same parameters to get the duration. [11]

We chose Google Maps, because it offers the functionalities that fulfil our system's needs. Google Maps is dynamic, and it takes into account the traffic, so the travel time estimations are realistic. [9]

4.2. PAYPAL

PayPal is an online payment system, which allows users to store money or transfer money through the system. Users can connect their credit cards to the service or transfer money to their account on PayPal. [12]

We decided to use PayPal as the method of payment for compensation since it is universally used and offers a simplistic way to implement payment process.

4.3. ALL RECEIPTS

All Receipts™ is application that collects the users' receipts. It also provides an interface through which requesting the receipts is possible.

5. DESCRIPTION OF INTERNAL COMPONENTS

5.1. CARBON FOOTPRINT CALCULATOR

Carbon footprint calculator is the component tasked with calculating the carbon footprint of the user using data from his or her receipts and car usage. It stores the calculated information to the database where it's usable for other components. It also has a basic exception handler that catches exceptions and handles them accordingly.

The car usage is calculated using location data from Google Maps and time from the device to get the distance and speed driven by car. It also checks if the speed has been high enough to assume the distance has been driven by car. The component then prompts a question to the user through API Gateway to make sure the user he/she drove a car. Then it calculates the CO2 emissions from the distance and speed. The car usage data is then sent to the database.

The other big part is to calculate the footprint from the receipt data. The footprint calculator gets the data from the API Gateway and uses also the car usage numbers to get the universal footprint as well as more specific product-based footprints. The footprint data is then stored to the database.

To accomplish these tasks, we divided the component to four classes that each have their own unique task. The GetLocationdataRequest class has a method which gets the data from Google Maps and returns it back to the main class. Get ReceiptdataRequest also invokes a request but for All Receipts service to get the receipt data as json.

The carUsage and CalculateFootprint classes are quite bit bigger and have more methods. CarUsage contains methods calculateSpeed and checkSpeed which are selfexplanitory and use the location data from the LocationdataRequest and time from the device internally. It also estimates CO2 usage from the distance and speed with calulateCO2 method. SendCarUsage method sends the data to our database.

The CalculateFootprint class contains methods for calculating the carbon footprint for common purposes with CalculateFootprint method, for each product with CalculateProductFP and weekly data with CalculateWeekly. The CalculateFootprint method uses the car data as well as the receipts data and calculates the whole carbon footprint for the day. CalculateProductFP calculates the carbon footprint for each product separately using the receipts data. CalulateWeekly takes the date and data that's been taken from the database using getDBData method and calculates the weekly footprint. SendFootprintdata stores the Footprint data to the database.

The code level split of the carbon footprint calculator to the four classes and the overarching main function made sense. The component needs to get and send data as well as make calculations on car usage and receipt data and all of them are easily arranged to their own classes. The main class is needed to call the different classes and keep the structure as well as call the exception handler.

5.2. CARBON FOOTPRINT FEEDBACK

Carbon footprint feedback is a component, that gives the user feedback based on their reported shopping and car usage habits. The component consists of 3 classes: FeedbackController, CarUsageFeedbackCreator and WeeklyFeedbackCreator. Feedback is given real-time and once a week.

Feedback creation is triggered by FeedbackController. It creates a class depending on the type of feedback. For weekly feedback, getWeeklyFeedback method is called with the past week's starting and ending date. For car usage feedback, getCarUsageFeedback is called with the user's route's end coordinates.

Real-time feedback is given every time the user reports car use. CarUsageFeedbackCreator first get the origin and destination coordinates with

setCoordinates method. Then createFeedback method is called, which calls queryRoute. The user's coordinates are used to query Google Maps Directions API, which returns public transportation's schedule for the route. The coordinates are also used to query Google Maps Duration Matrix API, which returns the duration for the routes. The responses from Google Maps are returned to createFeedback, which adds the user's car usage data to response. The class CarUsageFeedbackCreator is solely responsible for requesting schedules and duration from Google Maps and creating the feedback response and returning it in JSON form.

Weekly feedback is created by WeeklyFeedbackCreator. Feedback is created based on the user's consumption during the week. First, the week's starting and ending dates are set with setDate method. Then, createFeedback method is called which calls getMostConsumptiveItems. Most consumptive purchases are retrieved from the database with the dates in getMostConsumptiveItems. The items and their carbon footprints are returned in a dictionary and formatted to JSON in createFeedback.

When querying Google Maps Platforms, the endpoints for Directions API and Duration Matrix API are in appendices with examples.

Because sending feedback is a separate functionality in the application, it made sense to make its own component. Carbon footprint feedback system was divided to three classes: one controller and two classes for creating the feedback – a class for both types of feedback. If our system would give real-time feedback based on the receipts, then the component would have a separate class for that. The classes are quite small as the feedback doesn't need a lot of custom content.

5.3. COMPENSATION PAYMENT

Purpose of compensation payment component is for the user to be able to compensate for their carbon footprint by donating to organizations that help reduce CO2 emissions globally.

CompensationOptions class handles the information about different organizations. Information about organizations consists of the name of the organization, their home page URL, how much CO2 emissions can one compensate per euro and a PayPal ID.

PaymentRequest will create a PayPal donation button. It requires the amount of money that is being donated and the PayPal ID of the organization that the money is being donated to. Then it will create a donation button by using a script for it from Pay Pals API from which an example can be found in the appendices. This will also add functionalities to will happen on order and on approve. User pays and after the payment is successful

the StoreCompensation class is used to store information about the compensation in the database. [13]

This component was needed to be able to separate compensation functionality from rest of the functionalities. CompensationOptions class is necessary to handle the retrieval of different compensation options. It is also good to differentiate the PaymentRequest class since it is the part that takes care of producing the donation button for each instance of donating.

5.4. CONTROLLER

Controller is responsible for triggering calculations and feedback creation based on the user's actions. Controller consists of only one class.

When the user chooses to see weekly feedback, the Controller initializeWeeklyFeedback method triggers calculation in the Carbon Footprint Calculator and once the calculation is done and the results is stored, the Carbon footprint feedback is triggered to create the feedback and return it to the Controller, which returns it to the API.

Car usage feedback is triggered much like the weekly feedback, but with the initializeCarUsageFeedback method. The user gives confirmation, that they have used a car and that triggers the same kind of calculation and feedback creation workflow.

A controller was needed to organize workflow between the calculator and the feedback creator. Our controller solves the problem of triggering feedback, without adding more coupling between the calculator and the feedback creator.

6. SOURCES

[1] S. Brown. The C4 model for visualising software architecture. https://c4model.com/, read 9.4.2020

[2] Amazon Web Services. https://en.wikipedia.org/wiki/Amazon_Web_Services, read 9.4.2020

[3] What is AWS? https://aws.amazon.com/what-is-aws/, read 9.4.2020

[4] Amazon Cognito. https://aws.amazon.com/cognito/, read 9.4.2020

[5] AWS Amplify Authentication. https://aws-amplify.github.io/docs/js/authentication, read 9.4.2020

[6] Databases on AWS. https://aws.amazon.com/products/databases/, read 9.4.2020

[7] Amazon API Gateway. https://aws.amazon.com/api-gateway/, read 9.4.2020

[8] Serverless AWS Provider documentation.
https://serverless.com/framework/docs/providers/aws/, read 9.4.2020

[9] Google Maps Platform Overview, https://cloud.google.com/maps-platform, read 12.4.2020

[10] Google Maps Directions API,
https://developers.google.com/maps/documentation/directions/start, read 12.4.2020

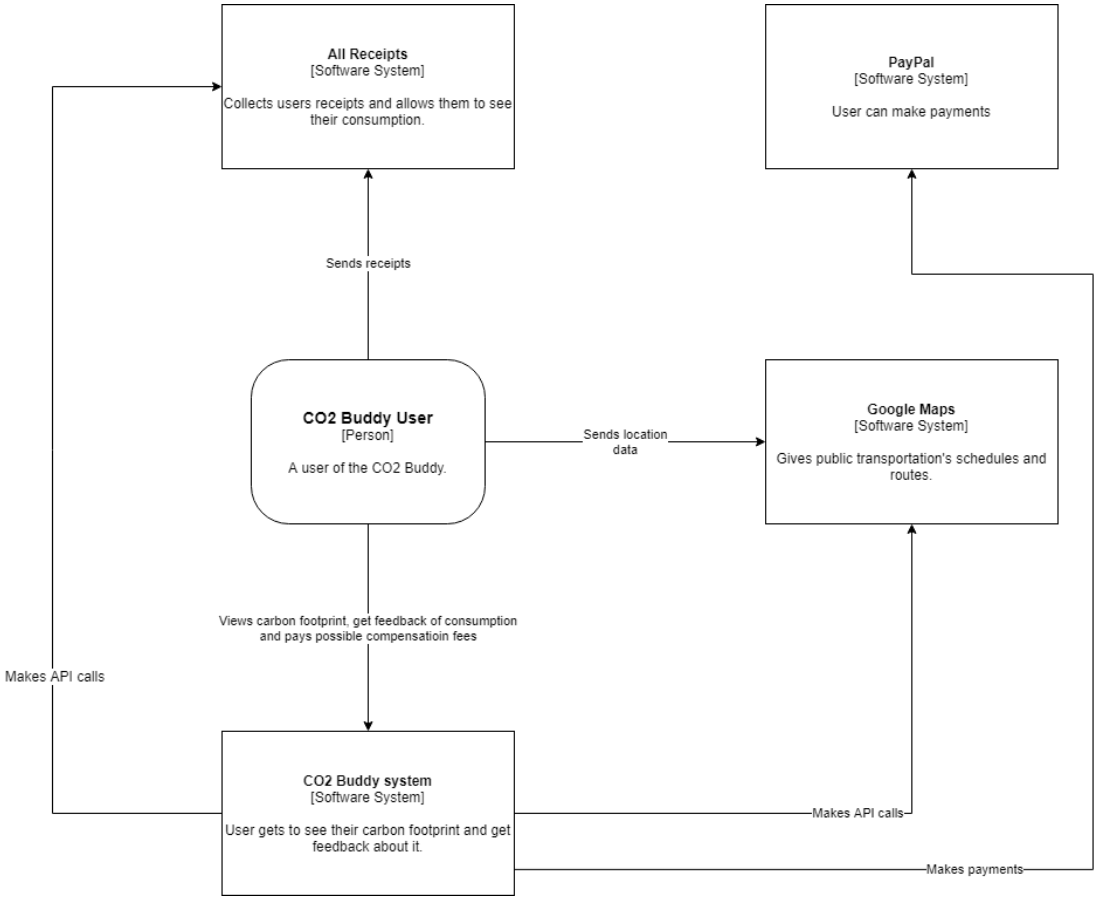
[11] Google Maps Duration Matrix API,
https://developers.google.com/maps/documentation/distance-matrix/intro, read 12.4.2020

[12] PayPal Wikipedia, https://en.wikipedia.org/wiki/PayPal, read 12.4.2020

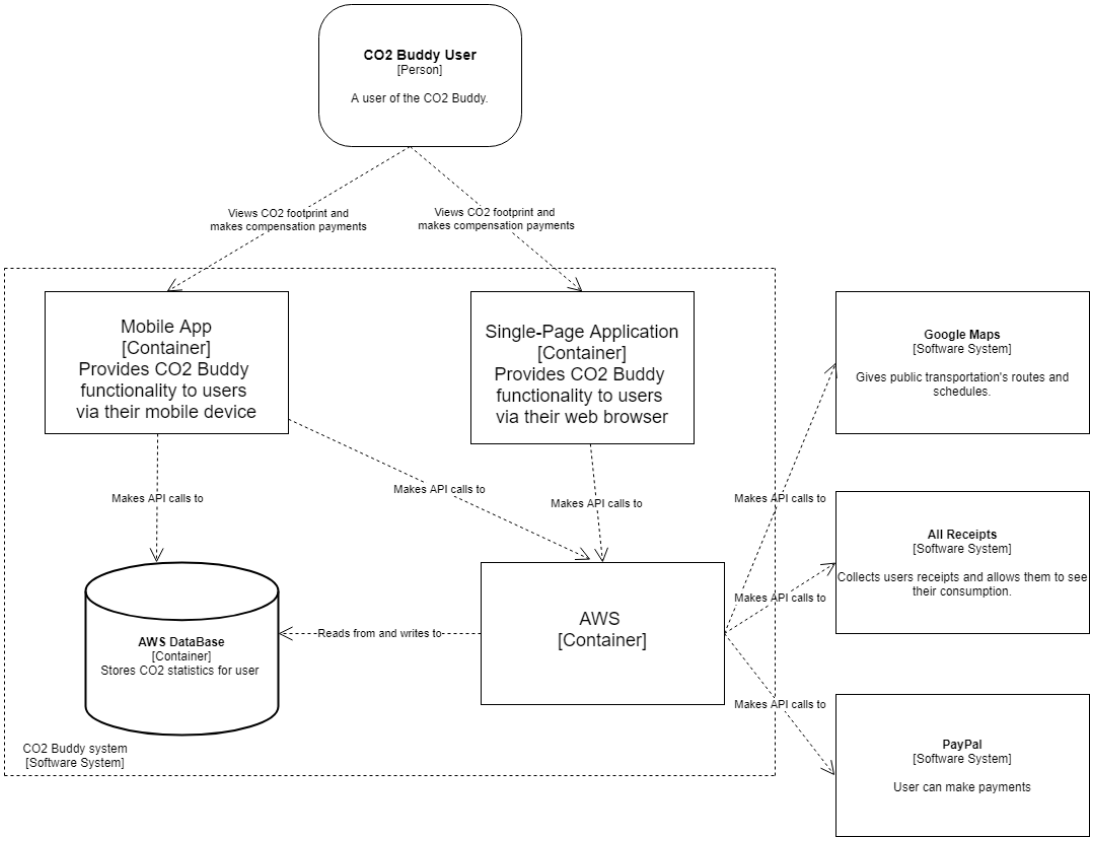
[13] PayPal Payment Buttons Integration ,
https://developer.paypal.com/demo/checkout/#/pattern/client, read 12.4.2020

7. APPENDICES

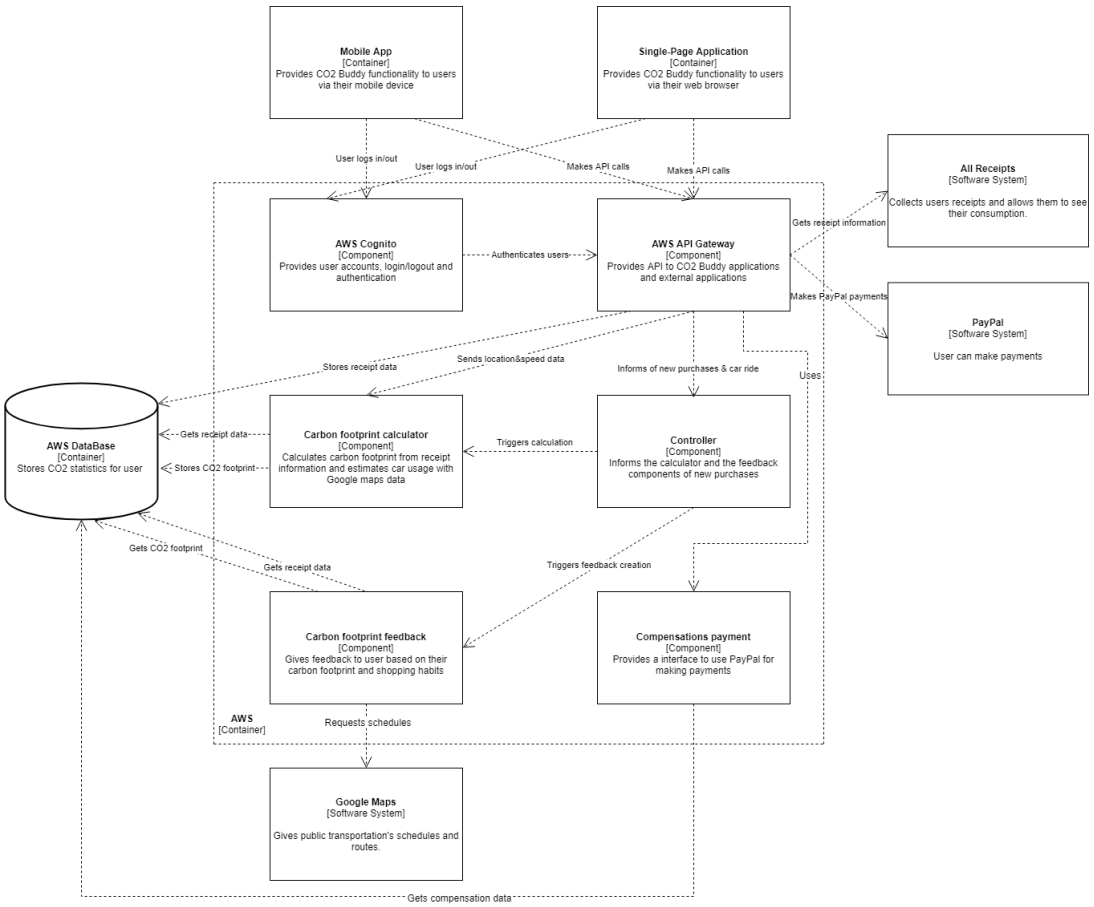
1. Context Diagram



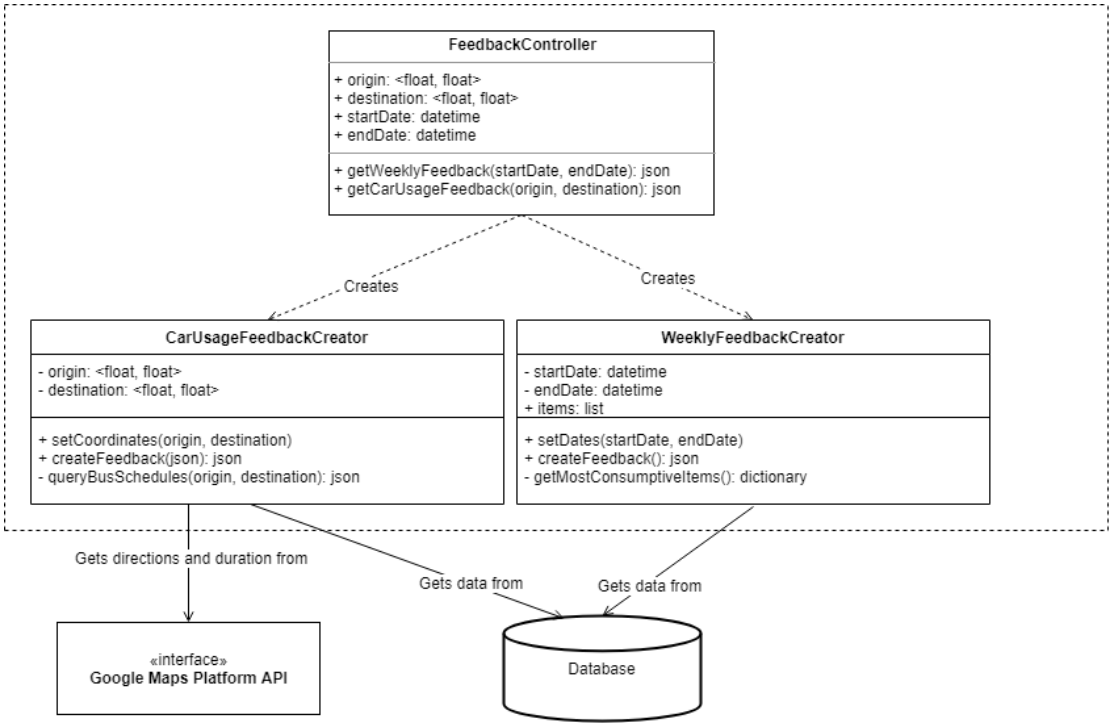
2. Container Diagram



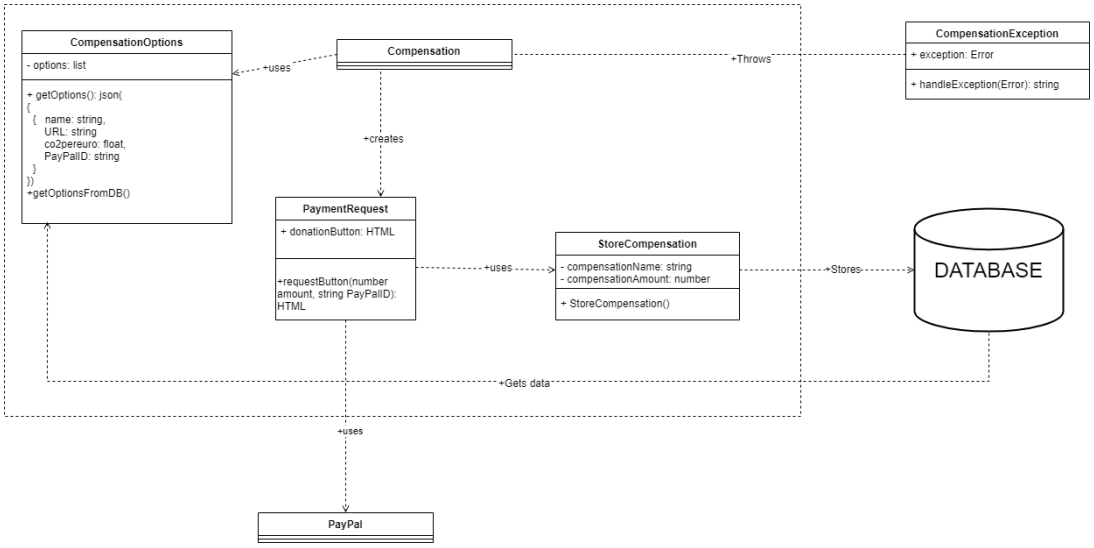
3. Component Diagram



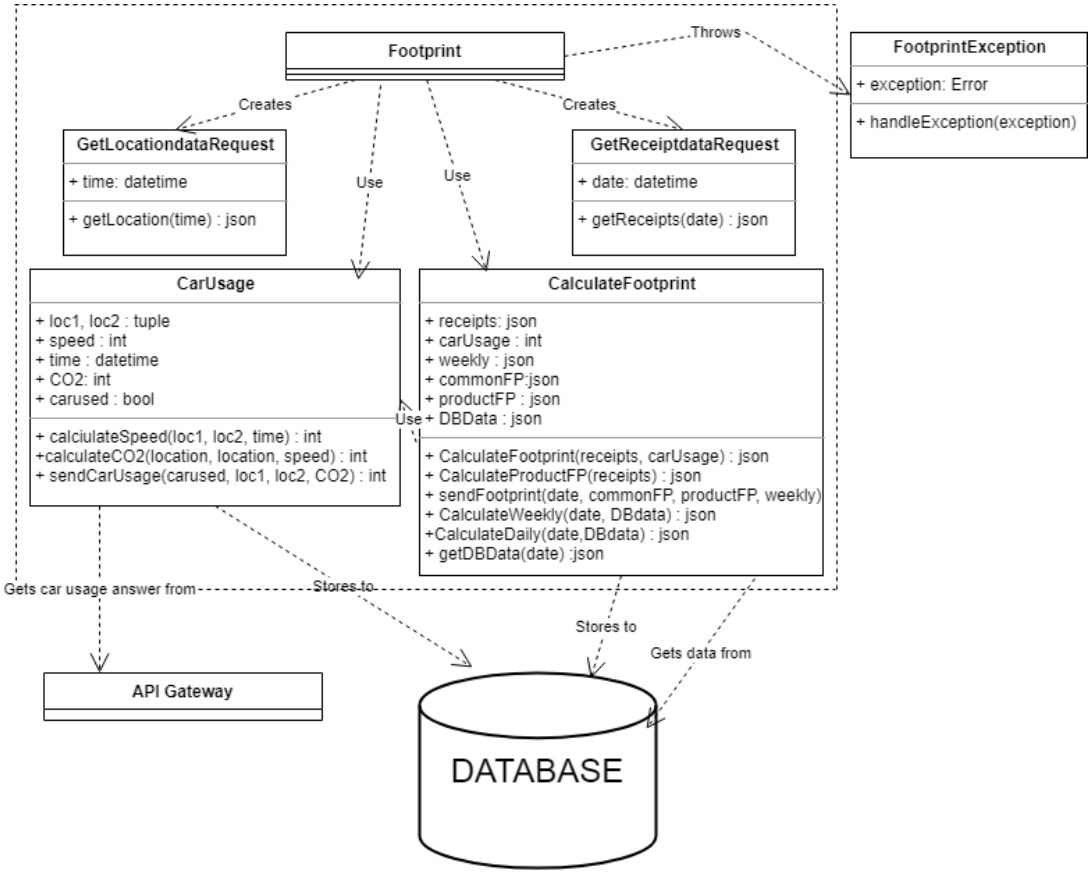
4. Carbon Footprint Feedback



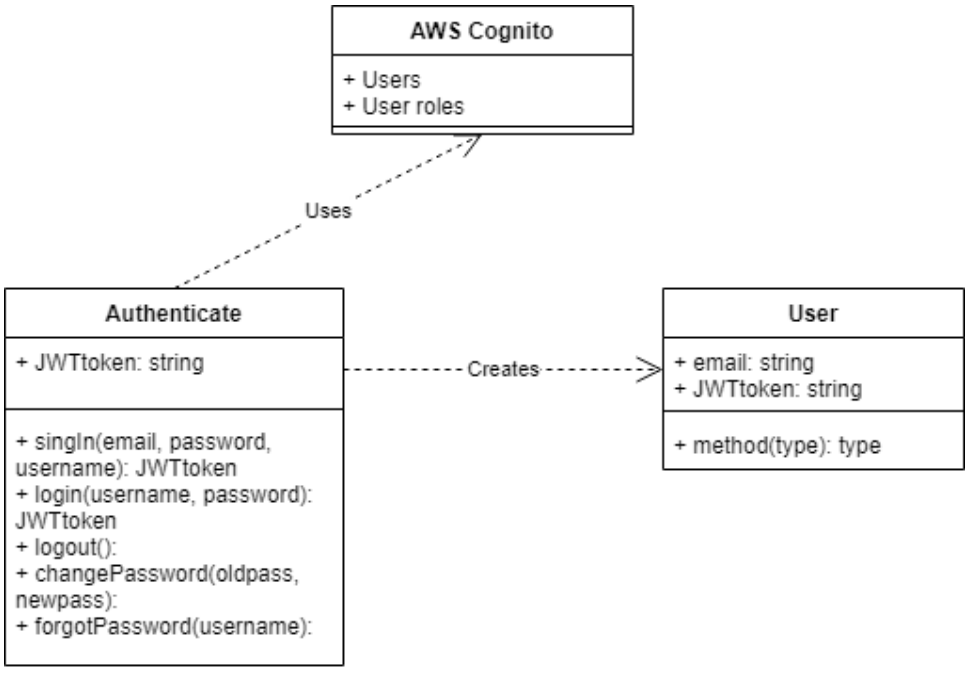
5. Compensations Payment



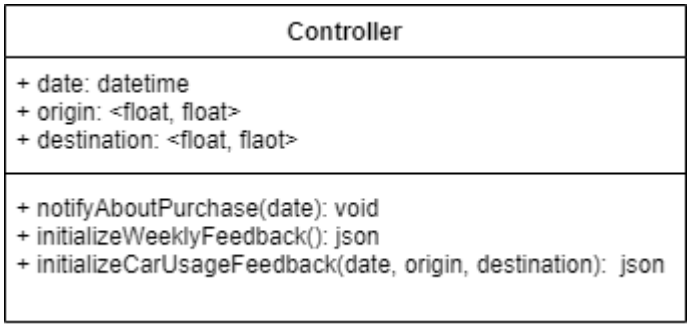
6. Carbon footprint calculator



7. AWS Cognito



8. Controller



9. Endpoints

   Google Maps Platform Directions API:

   https://maps.googleapis.com/maps/api/directions/outputFormat?parameters

   Example:

   https://maps.googleapis.com/maps/api/directions/json?origins=latitude,longitude&destinations=latitude,logitude&transit_mode=transit&key=API_key

   Google Maps Platforms Duration Matrix API:

   https://maps.googleapis.com/maps/api/distancematrix/outputFormat?parameters

   Example:

   https://maps.googleapis.com/maps/api/distancematrix/json?origins=latitute,logitude&destinations=latitude,longitude&transit_mode=transit&key=API_key

   Pay Pal button API:

   https://www.paypal.com/sdk/js?parameters

   Example:
   https://www.paypal.com/sdk/js?client-id=organization_id&currency=EUR

10. CO2 Buddy API

```
API:
swagger: "2.0"
info:
  title: CO2Buddy API
  version: 1.0.0

host: api.example.com
basePath: /v1
schemes:
  - https

paths:
  /carbon-footprint:
    get:
```

```
    summary: Returns whole carbon footprint of the user.
    produces:
      -  application/json
    responses:
      200:
        description: OK
/carbon-footprint/daily
  Get:
    Summary: Returns daily carbon footprint.
    Produces:
      -  application/json
    Responses:
      200:
        Description: OK
/carbon-footprint/weekly
  Get:
  Summary: Returns weekly carbon footprint.
  Produces:
    -  application/json
  Responses:
    200:
      Description: OK
/location
  post:
  Summary: Sends location data.
  parameters:
    - in: body
      name: location
      required: true
      schema:
        type: object
        properties:
          lat:
            Type: float
          lng:
            Type: float
  Produces:
    -  application/json
  Responses:
    200:
      Description: OK
/compensations
  Get:
  Summary: Gets a list of compensation options.
  parameters:
    - in: body
      name: compensation
      required: true
```

```
      schema:
        type: object
        properties:
          name:
            Type: float
          Url:
            Type: string
          Co2perEuro:
            Type: int
          PaypalAddress:
            Type: string
  Produces:
    -  application/json
  Responses:
    200:
      Description: OK
/donate
  Get:
  Summary: Creates a Pay Pal donation button.
  parameters:
    - in: body
      name: donation
      required: true
      schema:
        type: object
        properties:
          amount:
            Type: float
          PayPalID:
            Type: string
           Name:
             Type: string
  Produces:
    -  HTML
  Responses:
    200:
      Description: OK
/feedback
  Get:
  Summary: Get weekly feedbacks.
  parameters:
    - in: body
      name: feedback
      required: true
      schema:
        type: object
        properties:
          Item_name:
```

```
          Type: string
        Carbon_footprint:
          Type: float
Produces:
  -  application/json
Responses:
  200:
    Description: OK
```