Bilkent University

Department of Computer Engineering

Senior Design Project

SAVE: safe drive

Low Level Design Report

Elif Demir, Utku Kalkanlı, Büşra Ünver, Celal Bayraktar, Münevver Uslukılıç

Supervisor: Dr. Hamdi Dibeklioğlu

February 8, 2021

This report is submitted to the Department of Computer Engineering of Bilkent University in partial fulfillment of the requirements of the Senior Design Project course CS491/2.

1 Introduction

Everyday people use transportation to reach a destination. After the Covid-19 pandemic, people try to avoid using public transportation. As a result, private vehicle numbers in traffic increased significantly. More individual vehicles mean more errors in traffic. Most road accidents occur due to human error, and one of the most common types of error is driver fatigue. Traffic Accident Commission of Australia states that 20% of fatal road accidents involve driver fatigue. Furthermore, 30% of the severe crashes again involve driver fatigue [1]. Additionally, the driver's medical condition is also important in traffic. A study done by the Centre for Automotive Safety Research, University of Adelaide shows that 13% of the crashes investigated are caused by a medical condition [2].

Car companies supply some features in their cars to decrease such accidents caused by human error. However, the cost of those cars is not affordable in general and cars produced previously do not benefit from the latest developments in technology. Considering all these problems, we tried to come up with a solution to decrease human error factors in the traffic and to make the latest driver assistant features available for everyone. As a result, SAVE came up.

In the development process of the idea, we conduct market research on similar products on both vehicle and mobile application markets. After research, we find out there are similar products in the mobile application market like DRIvision [3] and car manufacturers already implement similar driver assistance applications on the vehicles. However, SAVE differs from these applications. Unlike DRIvision, SAVE mainly focuses on the driver along with the road related warnings. Although there are vehicles that provide similar features, we aim to supply these features at a much lower cost to drivers.

As mentioned before, the main focus of the SAVE will be on driver behavior. Additionally, we also provide other driver assistance like the crash emergency message and more. The features of the application will be explained in detail in the following sections.

1.1 Object design trade-offs

1.1.1 Usability vs. privacy

To use the functionalities the application offers, users should consent to record and store their faces when they're using the app. The stored images of the faces are used to track the features like closeness of the eye or the angle of the face. Storing the faces or the statistics about fatigue/tiredness on cloud could bring some privacy concerns for the users who are not comfortable with their personal information stored by an application, however majority of the smartphones have limited storage and

processors, allowing the application to run only a portion of the functionalities the application offers. Users can choose to not use those functionalities and not send their personal information to the cloud.

1.1.2 Space vs. Time

For some of the functionalities the application offers, the space of the smartphones starts to become insufficient. To avoid complications the application runs those functionalities like emotion detection or traffic light detection on the cloud. The communication with the cloud takes extra time, but the wide space cloud offers allows features to run smoothly.

1.2 Interface documentation guidelines

In the documentation we used a name form for classes like “ClassName”. Also, in functions and attributes we used similar form like “functionName” for function and “attributeName” for attributes. The class descriptions are made in the following form: First class name and description is made, following the class description there are attributes and lastly, after the class attributes, function names and descriptions of each of them given.

1.3 Engineering standards (e.g., UML and IEEE)

We used the IEEE format for referencing our resources used in reports and UML diagrams to design our application and classes.

1.4 Definitions, acronyms, and abbreviations

Remote Photoplethysmography: Remote PPG

Driver Assistance: Helping the driver of the vehicle by offering some useful features.

UML: Unified Modeling Language

Amazon Cloud Server: AWS

2. Packages

This section will demonstrate the general package diagram of the SAVE application. In order to achieve modularity, we wanted to divide our system into packages and classes inside those. We have indicated that we are going to use 3 tier architecture in our high level design in order to achieve the ability to upgrade any of these tiers independent from each other in order to respond to any changes.

Three tier architecture of our application contains presentation(user interface) tier, logic(controller) tier and data tier. These 3 tiers are developed and maintained independent from each other, while interacting and using each other's features.

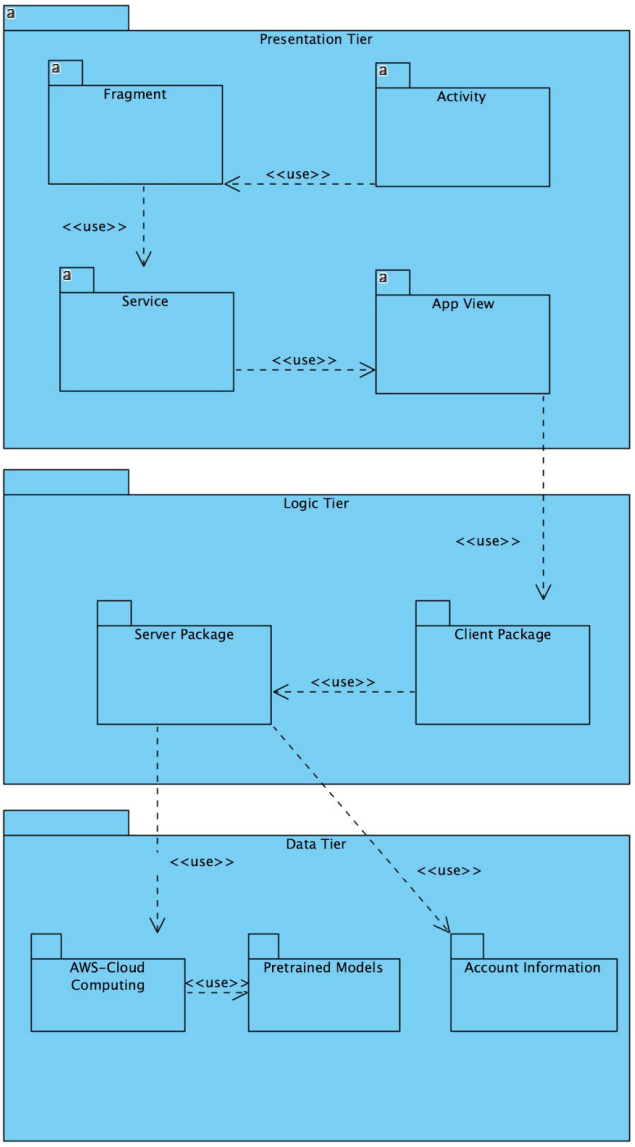

Figure 1. Package Diagram of SAVE

2.1 Presentation Tier Packages

Presentation tier is the top most level of our application. Purpose of this package is to display information and handle communication between the driver and the app. It displays obtained results with notifications as well as it interacts with the user via voice recognition during order taking and giving verbal outputs. The results it displays are determined by communication it does with other tiers of our application such as logic tier.

Design of this tier is aimed to have simple and useful user experience with minimum need to take input from the user as possible and have a satisfying ability to give information without distracting the user from the road.

2.1.1 Activity

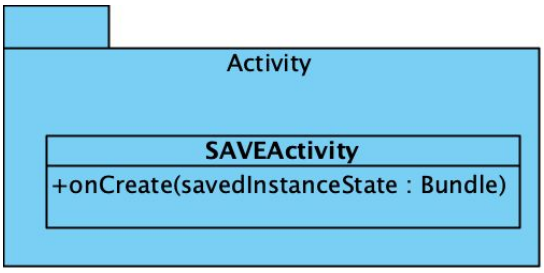

Figure 2.Package Diagram of Activity

Activity package contains the SAVEActivity class which is used to start the program. SAVEActivity class contains fragments that will load to screen when the app is opened.

2.1.2 Fragment

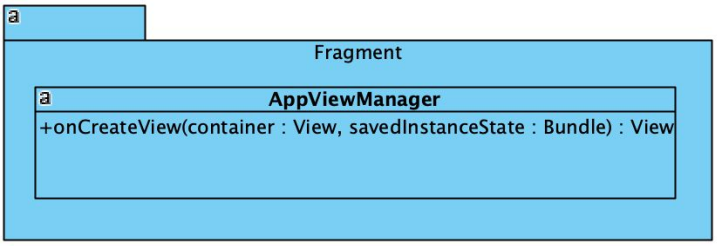

Figure 3. Package Diagram of Fragment

Fragment package contains an App View class which is used to show the App Window on top of the screen. Also this class interacts with the service package which is the intermediate manager of the system.

2.1.3 Service

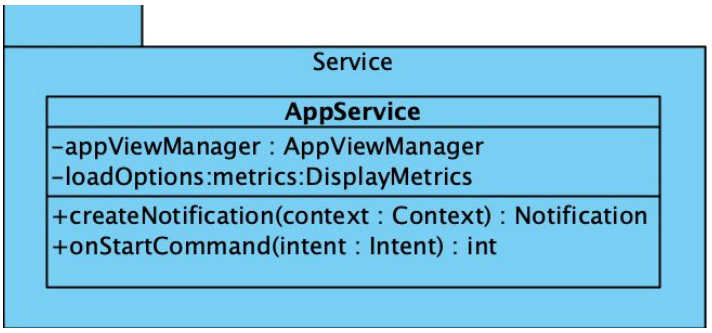

Figure 4. Package Diagram of Service

Service package contains the AppService class which has an AppViewManager instance. By this instance, an AppView can be controlled by using its functions. This class serves as an intermediate controller in the app.

2.1.4 App View

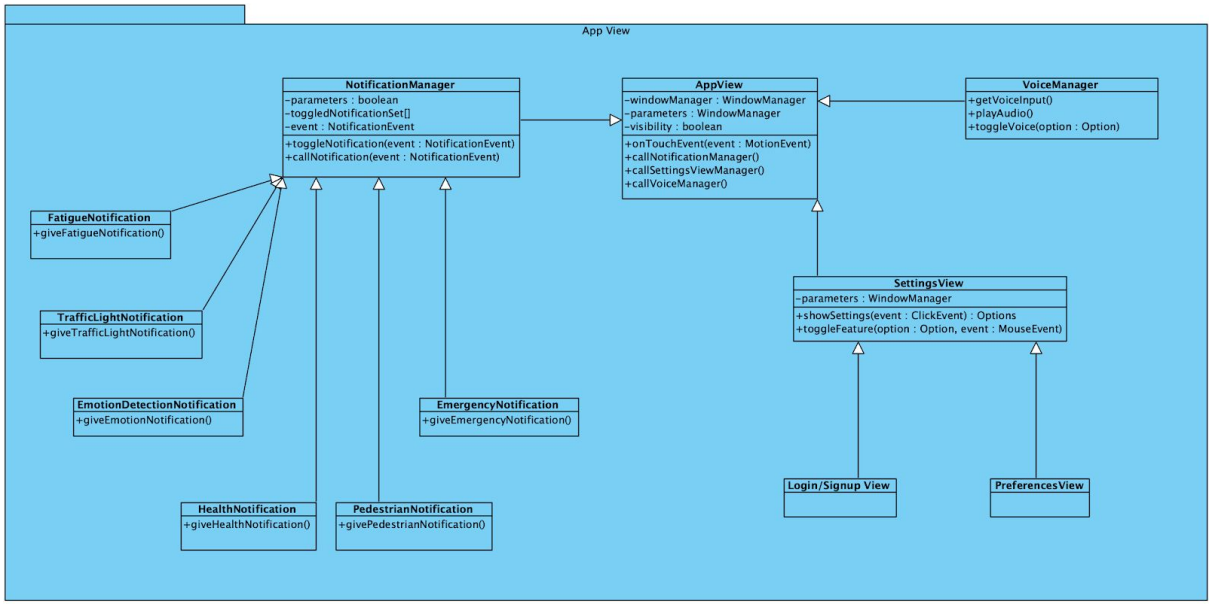

Figure 5. Package Diagram Flow of App View

The App View package contains the main presentation elements of the app which are AppView. It does the work of controlling VoiceManager functionality by taking voice input and playing voice output. It also toggles voice features on and off. NotificationManager class has the ability to control all notification functionalities as well as the SettingsView class has the ability to control view of the settings page.

2.2 Logic Tier Packages

2.2.1 Client Package

Logic tier client package is responsible for managing the functionalities that run on the client side of the app, those functionalities consist of face tracker, fatigue detection, voice manager, collusion detector, music player and cloud manager. Additional to the functionalities of the class all classes have getter and setter functions for their attributes.

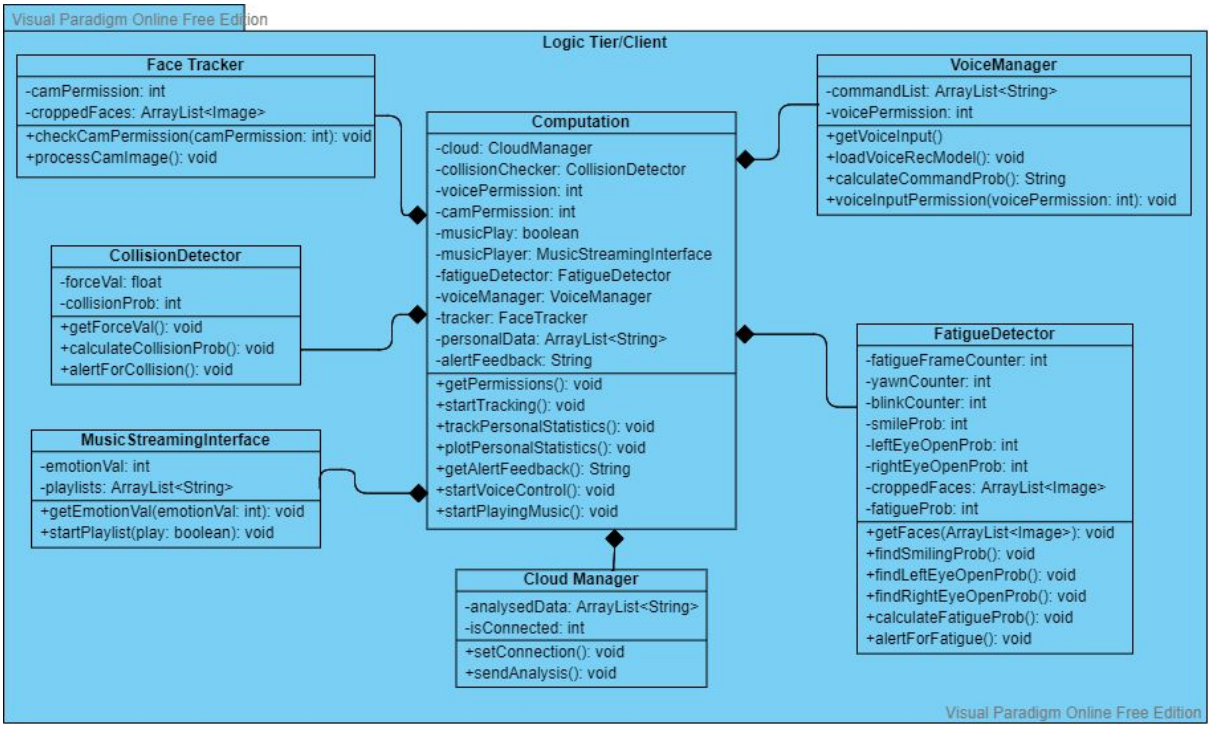

Figure 6. Package Diagram of Logic Tier/Client

2.2.2 Server Package

Logic tier server packages are responsible for functionalities that run on the cloud space, those functionalities include clientManager, TrafficLightDetector, EmotionDetector, HealthConditionTracker. Additional to the functions listed each class has it's getter and setter methods for their attributes.

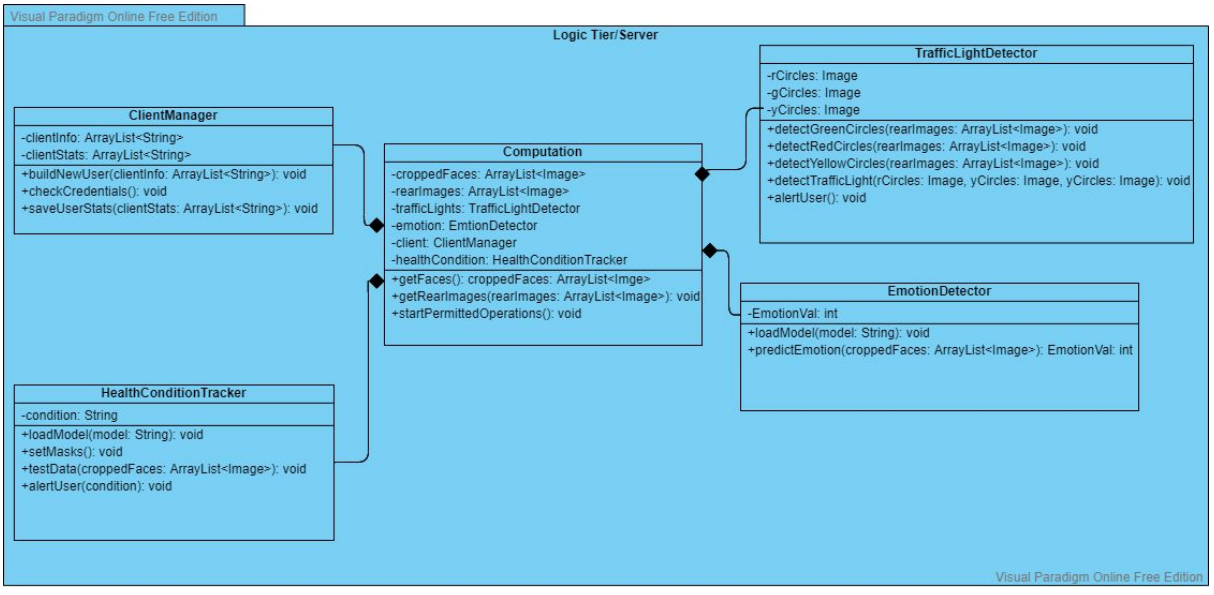

Figure 7. Package Diagram of Logic Tier/Server

2.3 Data Tier Packages

Data tier keeps the data that is needed for our application. Those data are accessible to other subsystem tiers for their operations. Data tier includes a data access layer that encapsulates the mechanism and exposes the data. This layer provides functionalities to the logic tier that requires pretrained models and user data to work its functionalities fully.

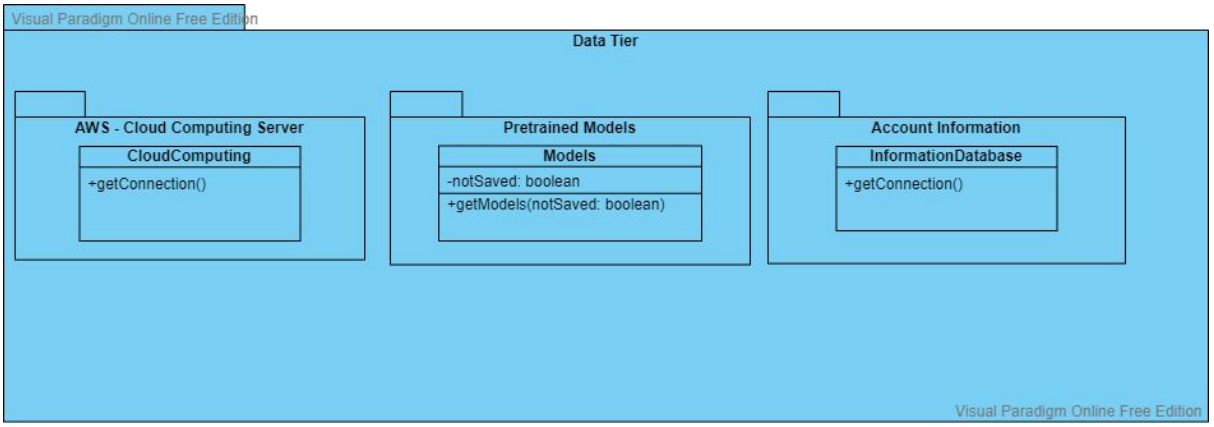

Figure 8. Package Diagram of Data Tier

3. Class Interfaces

3.1 Presentation Tier Classes

3.1.1 Activity Package Classes

Class: SAVEActivity
This class is the main class that starts the application and connects other parts of the application
Attributes:
-
Functions:
onCreate(): opens the application in sign in interface if the sign in part have not been done; otherwise opens the home screen of the application

3.1.2 Fragment Package Classes

Class: AppViewManager
This class manages the running app in terms of notifications, current view, and sound control
Attributes:

Functions:
onCreateView(container : View, savedInstanceState : Bundle) : *creates the current view with respect to the saved instance state*

3.1.3 App View Classes

Class: NotificationManager
This class manages the incoming notifications from other classes such as EmotionDetectionNotification
Attributes:
• parameters: boolean • toggledNotificationSet[] • event: NotificationEvent
Functions:
toggleNotification(event: NotificationEvent): *sets notification settings on and off* callNotification(event: NotificationEvent): *initializes one of the subclasses to give notification*

Class: FatigueNotification
This class sends notification when fatigue is detected
Attributes:
Functions:
giveFatigueNotification(): *gives fatigue notification to user*

Class: TrafficLightNotification
This class sends notification about the condition of traffic light that is detected from the camera
Attributes:
Functions:
giveTrafficLightNotification() *gives notification on traffic light condition to user*

Class: EmotionDetectionNotification
This class gives notifications related to the driver' emotional condition in a general way

Attributes:
Functions:
giveEmotionNotification() *sends notifications based on emotion of the user*

Class: Health Notification
This class gives notification on the driver's health condition such as heart rate
Attributes:
Functions:
giveHealthNotification(): *gives notification about pulse rate and health to the user*

Class: Pedestrian Notification
This class gives notification if a pedestrian is on the road and creates danger
Attributes:
Functions:
givePedestrianNotification(): *gives pedestrian warning notification to user*

Class: Emergency Notification
This class gives notification when there is a sudden emergency is detected such as crash
Attributes:
Functions:
giveEmergencyNotification(): *gives emergency situation notification to user*

Class: AppView
This class connects notifications, settings and voice managers and provides the functional part of the application in usage. It is the main element of the app which is displayed on the screen
Attributes:
• windowManager: WindowManager • parameters: WindowManager • visibility: boolean
Functions:

onTouchEvent(event: MotionEvent) *determines what should happen in a certain type of interaction happens with the user* callNotificationManager() *AppView calls notification manager class in order to maintain notification activities.* callSettingsViewManager() *AppView calls settings view class in order to show settings features.* callVoiceManager() *AppView calls VoiceManager class in order to use voice input and output features.*

Class: SettingsView
This class make the settings options visible to enabling and disabling
Attributes:
• parameters: WindowManager
Functions:
showSettings(event: ClickEvent) *shows settings* toggleFeature(option: Options, event: MouseEvent) *toggles a feature's state*

Class: Login/SignUpView
Shows login and sign up
Attributes:
Functions:

Class: PreferencesView
Shows the preferences to the user
Attributes:
Functions:

Class: VoiceManager
Voice manager controls the verbal communication between the app and the user.
Attributes:
Functions:

getVoiceInput(event: MotionEvent) *gets users commands* playAudio(event: MotionEvent) *plays an audio message to inform user* toggleVoice(option: Option, event MouseEvent) *sets voice feature on or off*

3.1.4 Service Classes

Class: AppService
This is an intermediate controller class and it connect the presentation tier and logic tier
Attributes:
● appViewManager: AppViewManager ● loadOptions: metrics: DisplayMetrics
Functions:
createNotification(context: Context) : Notification *creates notification* onStartCommand(intent: Intent) : int *Includes some rules about the application that will be needed when the app starts*

3.2 Logic Tier Package Classes

3.2.1 Logic Tier / Client Classes

Class: CollisionDetector
This class is responsible for detecting a collision in case of an accident.
Attributes
● float forceVal ● int collisionProb
Functions
void getForceVal(): gives the exposed G Force value *void calculateCollisionProb():* according to calculated G Force value calculates collision probability. *void alertForCollision():* collision probability higher than threshold value method gives an alert.

Class: MusicStreamingInterface
This class is responsible for keeping the driver alive or decreasing the level of stress by playing music.
Attributes

- int emotionVal
- ArrayList<String> playlists

Functions

void getEmotionVal(int emotionVal): gives the emotion value.
void startPlaylist(boolean play): according to emotion value starts a playlist.

Class: Face Tracker

This class is responsible for tracking the face of the driver.

Attributes

- int camPermission
- ArrayList<Image> croppedFaces

Functions

void checkCamPermission(int camPermission): Checks if user give permission to access the camera.
void processCamImage(): Crop and normalize the read image to be processed by the convolutional layer models.

Class: FatigueDetector

This class is responsible for detecting driver fatigue.

Attributes

- int fatigueFrameCounter
- int yawnCounter
- int blinkCounter
- int smileProb
- int leftEyeOpenProb
- int rightEyeOpenProb
- ArrayList<Image> croppedFaces
- int fatigueProb

Functions

void getFaces(ArrayList<Image> croppedFaces): gives the cropped faces.
void findSmilingProb(): calculates the smiling probability.
void findLeftEyeOpenProb(): calculates the probability of the left eye being open.
void findRightEyeOpenProb(): calculates the probability of the right eye being open.
void calculateFatigueProb(): calculates the fatigue probability according to left and right eye probability.
void alertForFatigue(): alerts the user if fatigue probability is higher than threshold value.

Class: CloudManager

This class is responsible for communicating with cloud server.

Attributes

- ArrayList<String> analysedData
- int isConnected

Functions

void setConnection(): Connects to the cloud server.
void sendAnalysis(): Save the tracked statistics of the user to the cloud for plotting the statistics.

Class: VoiceManager

This class is responsible for getting voice commands from the user.

Attributes

- ArrayList<String> commandList
- int voicePermission

Functions

void getVoiceInput(): gets the voice input.
void loadVoiceRecModel(): loads the voice recognition model.
String calculateCommandProb(): calculates the probability of a given command.
void voiceInputPermission(int voicePermission): checks if user gives the permission for access microphone.

Class: Computation

This class uses the instances and functionalities of other classes in the package to run them simultaneously and track overall usage statistics to plot the past data.

Attributes

- CloudManager cloud
- CollisionDetector collisionChecker
- int voicePermission
- int camPermission
- boolean musicPlay
- MusicStreamingInterface musicPlayer
- FatigueDetector fatigueDetector
- VoiceManager voiceManager
- FaceTracker tracker
- ArrayList<String> personalData
- String alertFeedback

Functions

void getPermissions(): gives all permissions taken from the user.
void startTracking(): starts to face tracking process.
void trackPersonalStatistics(): tracks the users statistics over the time.
void plotPersonalStatistics(): visualize the tracked statistics.

void getAlertFeedback(): Feedback from the user for a specific occurrence like fatigue detection or health condition problem. *void startVoiceControl():* starts the voice command process. *void startPlayingMusic():* starts playing music.

3.2.2 Logic Tier / Server Classes

The data received from the client phone to be processed on the server is sent into Data Tier to be computed. This package handles the models that are going to be in the connection with the cloud servers and the images for the computing.

Class: ClientManager
This class is responsible for building new clients and saving the statistics of the user.
Attributes:
● private ArrayList<String> clientInfo ● private ArrayList<String> clientStats
Functions:
public void buildNewUser(ArrayList<String> clientInfo): builds new user and saves the information from clientInfo list. **public void checkCredentials():** checks if the user fills the required information for the account correct. **public void saveUserStats(ArrayList<String> clientStats):** saves the user statistics with using the clientStats list.

Class: HealthConditionTracker
This class is responsible for tracking the health condition.
Attributes:
● private String condition
Functions:
public void loadModel(String model): loads the pre trained model that executes the health condition tracking. **public void testData(ArrayList<Image> croppedFaces):** tests the data using the frames of the croppedFaces coming in the list by sending the HTTP requests to the cloud servers for the frames. **public void alertUser(condition):** alerts the user if any unhealthy situation is detected.

Class: Computation

This class is responsible for the required information, frames and models for application to do its work. It uses front and rear cameras and connects the models that are going to be executed with cloud servers to the application.

Attributes:

- private ArrayList<Image> croppedFaces:
- private ArrayList<Image> rearImages:
- private TrafficLightDetector trafficLights:
- private EmotionDetector emotion:
- private ClientManager client:
- private HealthConditionTracker healthCondition:

Functions:

public getFaces(ArrayList<Image> croppedFaces): detects the faces comes from the front camera.

public void getRearImages(ArrayList<Image> rearImages): detects the traffic lights from the frames coming from the rear camera.

public void startPermittedOperations(): sends the required frames and information to the parts of the application according to the permissions of the user that are saved in the settings.

Class: TrafficLightDetector

This class is responsible for tracking the changes in light color when any traffic light is detected.

Attributes:

- private Image rCircles
- private Image gCircles
- private Image yCircles

Functions:

public void detectGreenCircles(ArrayList<Image> rearImages): detects the green light circles by using the images coming from the rear camera.

public void detectRedCircles(ArrayList<Image> rearImages): detects the red light circles by using the images coming from the rear camera.

public void detectYellowCircles(ArrayList<Image> rearImages): detects the yellow light circles by using the images coming from the rear camera.

public void detectTrafficLight(Image rCircles, Image yCircles, Image gCircles): detects if there are any traffic lights in the rear view by using the read, yellow, green circles in the image.

public void alertUser(): alerts the user when the light color changes from red to the green.

Class: EmotionDetector
This class is responsible for detecting the negative or positive emotions of the user
Attributes:
● private int EmotionVal
Functions:
public void loadModel(String model): loads the pre trained model that executes the detection of the negative or positive emotions. **public int predictEmotion(ArrayList<Image> croppedFaces):** predicts emotion with sending HTTP requests to the server using the croppedFace lists.

3.3 Data Tier Package Classes

This package provides the connection between the application and the cloud servers and the database to save the statistics of the user. Other than that, it also stores the pre trained models.

Class: CloudComputing
This class is responsible for connecting the application to the cloud server.
Attributes:
Functions:
public getConnection(): connects the application to the Amazon Web Services in order to use the EC2 computing service.

Class: Models
This class is responsible for saving models.
Attributes:
● private boolean notSaved
Functions:
public void getModels(boolean notSaved): loads the models to the application.

Class: InformationDatabase

This class is responsible for connecting to the database in order to save the statistics of their fatigue, health and emotional conditions.
Attributes:
• public void getConnection():
Functions:
public void loadModel(String model): loads models and saves the statistics of the user in the settings.

4. Glossary

Remote Photoplethysmography	Remote Photoplethysmography (PPG) is a contactless way to measure human cardiovascular activity by measuring the reflection variations of the skin registered by a video camera [4]. It only requires video recording with a high-resolution camera[5].
Eye aspect ratio	Eye aspect ratio process is the combination of eye localization, analyzing the whites of eyes and determining the period of time that white region of the eye disappears to indicate the blink rate of the human by using facial landmarks[6].
Mouth aspect ratio	Mouth aspect ratio The horizontal and vertical distance of the mouth by using 2D facial landmark locations [7].

5. References

[1] *Fatigue statistics*, Transport Accident Commision. Accessed on: Nov. 21 2020. [Online]. Available: https://www.tac.vic.gov.au/road-safety/statistics/summaries/fatigue-statistics

[2] T. Lindsay, "Medical conditions as a contributing factor in crash causation." ResearchGate, May, 2018.

[3] *DRIvision*. Accessed on: Nov. 21 2020. [Online]. Available: https://drivision.wordpress.com/

[4] Verkruysse W., Svaasand L. O., Nelson J. S., "Remote plethysmographic imaging using ambient light," Opt. Express 16(26), 21434–21445 (2008).10.1364/OE.16.021434

[5] *What is RPPG?*, Noldus. 2020. Accessed on: Nov. 21 2020. [Online]. Available: https://www.noldus.com/blog/what-is-rppg

[6] Rosebrock, A., 2020. *Eye Blink Detection With Opencv, Python, And Dlib - Pyimagesearch*. PyImageSearch. Accessed on: Nov. 21 2020. [Online]. Available: https://www.pyimagesearch.com/2017/04/24/eye-blink-detection-opencv-python-dlib/

[7] Kir Savaş, Burcu & Becerkli, Yaşar. (2018). Real Time Driver Fatigue Detection Based on SVM Algorithm. 1-4. 10.1109/CEIT.2018.8751886.