Bilkent University
Department of Computer Engineering

Senior Design Project

FAVEO

Low Level Design Report

Zafer Çınar 21601514
Engin Deniz Kopan 21301826
Enes Varol 21604086
Enes Yıldırım 21602725

Supervisor: Uğur Doğrusöz
Jury Members: Çiğdem Gündüz Demir, Hamdi Dibeklioğlu

February 17, 2020

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

284 million visually impaired people are living around the world[1] and there are 220 thousand visually impaired living in Turkey [2]. They can't benefit from technology as much as others.FAVEO is an online quiz application for android that utilizes text to speech [3], Voice Control [4], TalkBack [5] and image processing to help visually impaired people's studying. In more detail they can solve questions, test their knowledge and also prepare their questions and share with others.

Application's main features will be solving questions as well as preparing questions. User will choose a topic to solve questions and they can solve with the help of TalkBack or Voice Control. Questions will be read by text to speech. To upload a question user can use upload image option that extracts the text from image and convert it to question. User can make changes on result of image processing or they can upload a voice questions.If they like parents can create custom quiz which will be accessible only by their account and can be modified.

Briefly, FAVEO is android application to help visually impaired people's studying. It is everybody's right to self-improve themselves yet unfortunately some disabled people have much less resources in order to self educate or self challenge. With Faveo, we are aiming to create a friendly environment that challenges the player while still being simple to operate.

1.1 Object Design Trade-offs

1.1.1 Cost vs Performance

FAVEO will provide questions to solve and image to question service for customers. However, good database and cloud application platforms are requires paid subscriptions. Since we have limited budget FAVEO will have a trade-off between cost and accuracy.

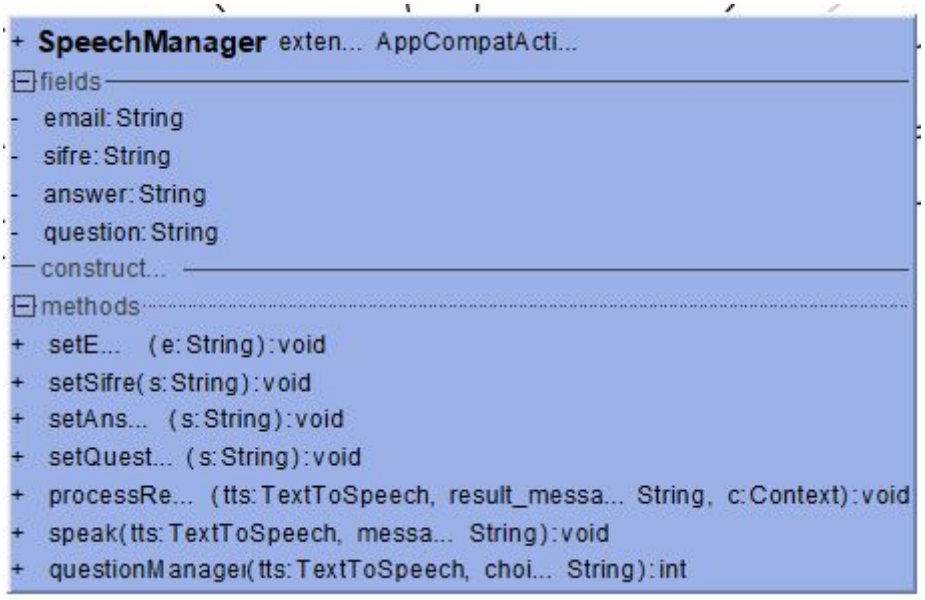
1.1.2 Scalability vs Performance

Serving multiple customers is main priority of FAVEO. However, we also want to consider performance with high importance. As conclusion, serving multiple customers with high performance can be challenging.

1.1.3 Usability vs Functionality

Faveo aims to help visually impaired people's education. Therefore, the system should be sufficiently useful for them. Customer friendly interface will affect the variety of the functionalities.

1.2 Interface Documentation Guidelines



Access modifiers will be denoted by `+`,` -', '#':

- + : public
- - : private
- # : protected

1.3 Engineering Standards

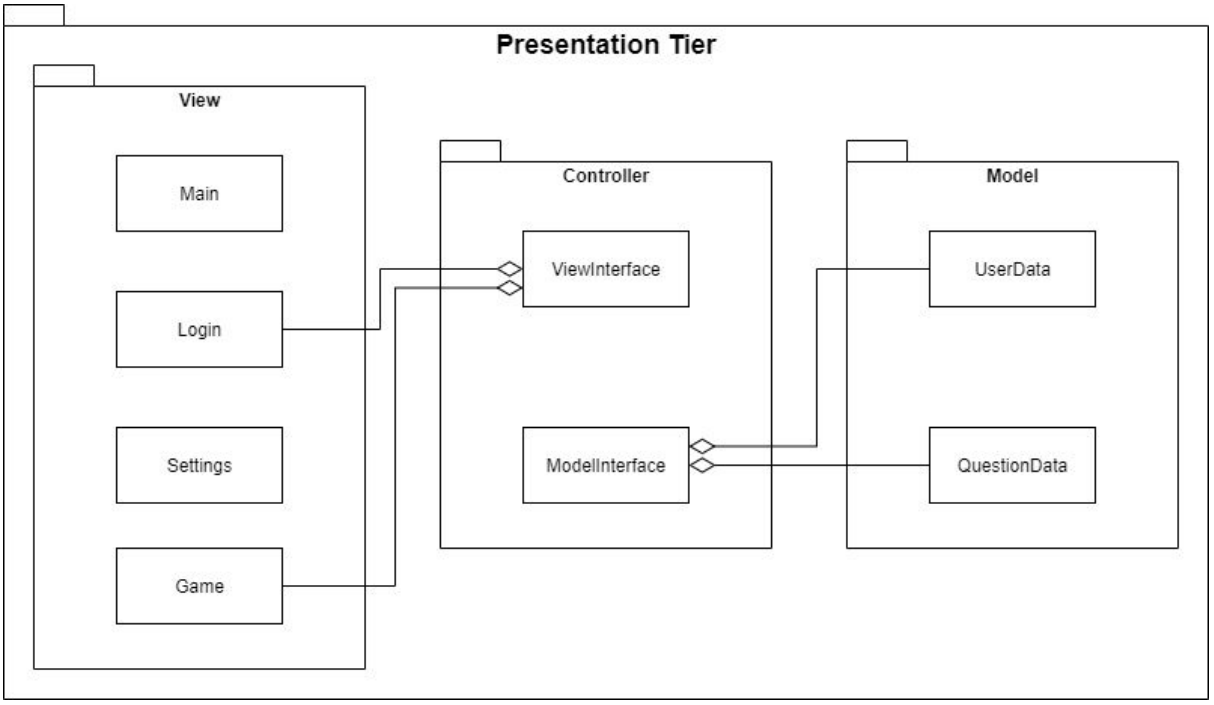
The diagrams in this report follow standard Unified Modeling Language (UML). On the other hand, to quote the references, we referred to guides from The Institute of Electrical and Electronics Engineers (IEEE).

1.4 Definitions, Acronyms and Abbreviations

- **API:** Application programming interface.
- **REST:** REpresentational State Transfer
- **UML:** Unified Modeling Language.

2. Packages

2.1 Presentation Tier



This tier is also known as client tier. The top-most level of the application is the user interface. The main function of the interface is to translate tasks and results into something that the user can understand easily.

2.1.1 View

View Class is responsible for the interaction between the user and application. All the visuals and interfaces are managed by View Class along with functional menus and settings.

Login: User interface for registering or logging in the application.
Settings: The interface for accessing various settings along with the accessibility options.
Main: The general interface of the menus how their transitions.
Game: The interface to be used when a game is on the screen of the application.

2.1.2 Controller

Controller Class creates the communication between the View Class and the backend of the application. Both Model and View Classes connect to the Controller Class which then provides the required traffic both ways.

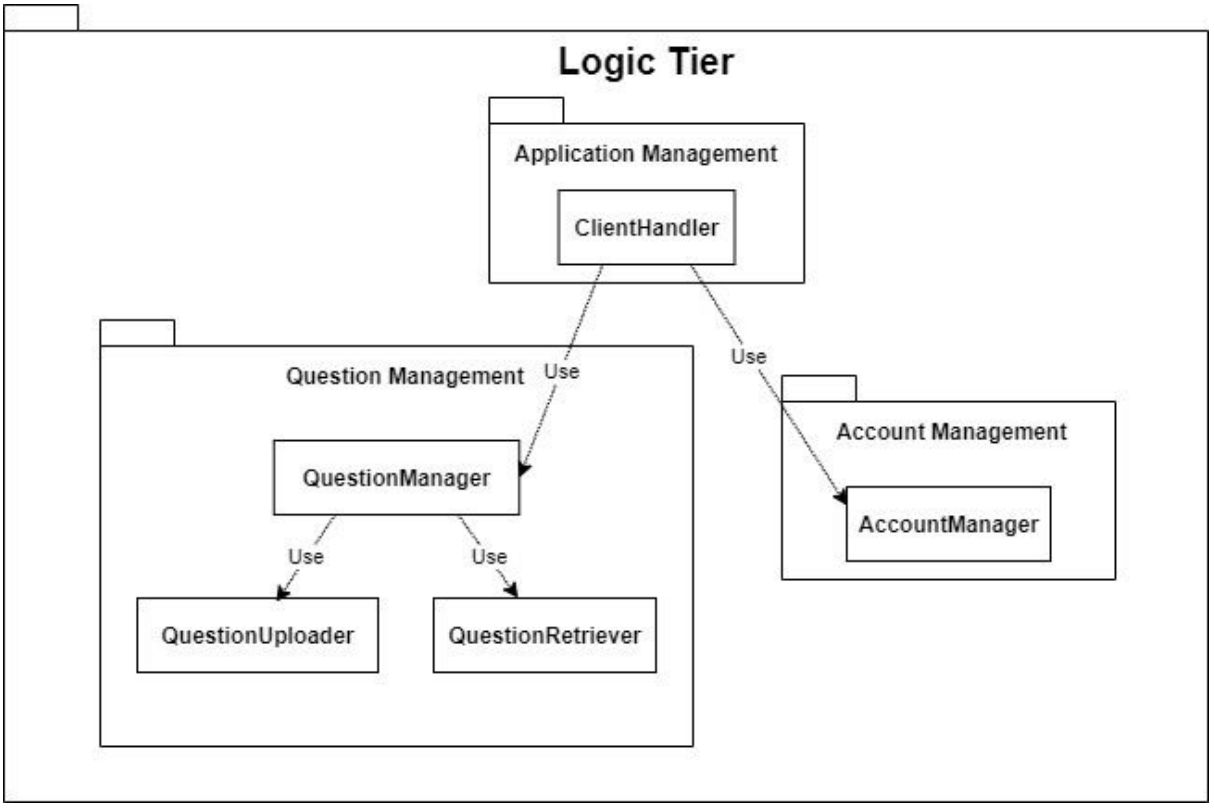
ViewInterface: Creates the interface between the View Class and and Controller Class.
ModelInterface: Creates the interface between the Model Class and and Controller Class.

2.1.3 Model

Model Class manages the questions and other various data which is relevant for the interface. This data might belong to the user or the database where the questions are stored.

UserData: Manages the preferences and identification of users.
QuestionData: Manages the questions, types of questions and the media that comes with corresponding questions.

2.2 Logic Tier



This tier is responsible for the main functionality of the system. It serves as the connection between the presentation tier and data tier. Transporting the information between the user and the database for activities such as uploading the question, retrieving a question and account management happens in this tier.

2.2.1 Application Management

This package is responsible for directing the application according to the information received from the user from the presentation tier.

ClientHandler: It handles the requests made by the user.

2.2.2 Question Management

Question Management is responsible for handling the activities involving the questions such as answering a question, uploading a question or retrieving a question.

QuestionManager: It decides which operation will be performed based on the information received from the user.

QuestionUploader: It communicates with the data tier to upload a question to the database.

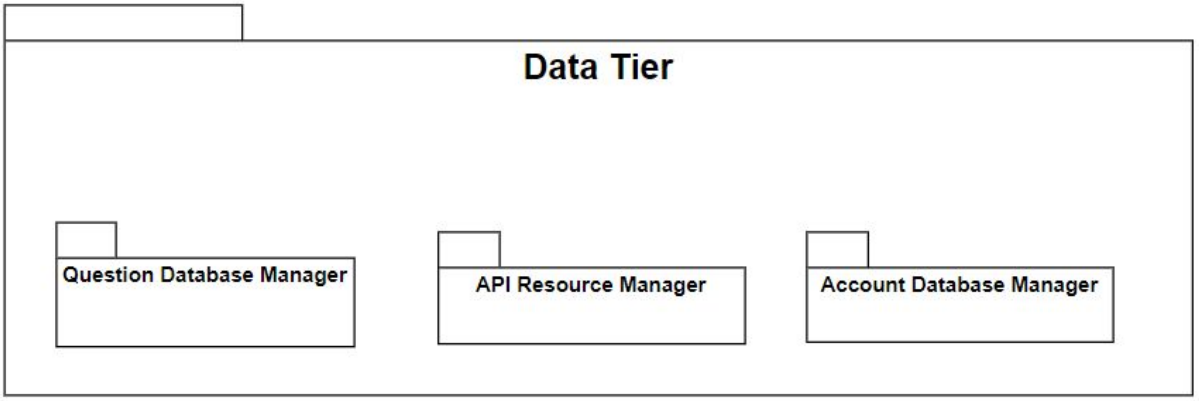
QuestionRetriever: It communicates with the data tier to retrieve a question specified with the given category and difficulty.

2.2.3 Account Management

Account Management is responsible for the operations that involves the user accounts.

AccountManager: It performs the activities such as logging in, creating account or logging out.

2.3 Data Tier



In this tier, information is stored and retrieved from a database system. The information is then passed back to the logic tier for processing, and then eventually back to the user. It is going to be deployed on server side. Providing secure and reliable access to the database is the main functionality of this layer.

2.3.1 Question Database Management

Question Database Manager is responsible for storing quiz questions and responding to calls coming from logic tier.

2.3.2 API Resource Management

API Resource Manager is responsible for retrieving the text which will be captured from a photo and storing it. It is written as RestAPI.

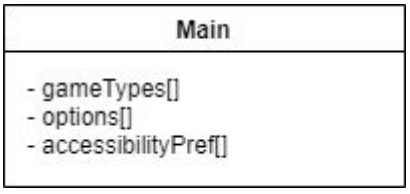
2.3.3 Account Database Management

Account Database Management is responsible for managing the user profile information.

3. Class Interfaces

3.1 Presentation Tier Interfaces

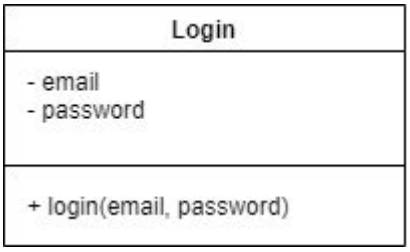
3.1.1 Main



Attributes:

- **gameTypes[]:** keeps the types of game modes.
- **options[]:** keeps the other menu items such as settings and uploading question.
- **accessibilityPref[]:** keeps what the user prefers in terms of accessibility in relation to the main menu.

3.1.2 Login



Attributes:

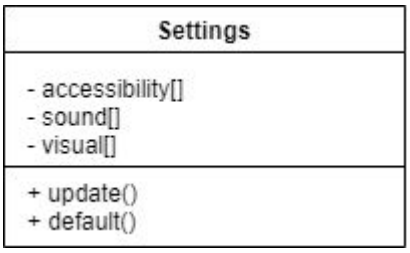
- **email**: keeps the email of the user.

- **password:** keeps the password of the user.

Methods

- **login(email, password):** attempts to log the user in.

3.1.3 Settings



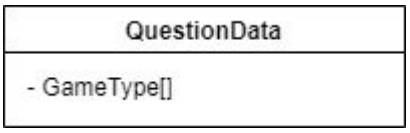
Attributes:

- **accessibility[]:** keeps the accessibility options for the user.
- **sound[]:** keeps the options for sound.
- **visual[]:** keeps the options for visuals.

Methods:

- **update():** updates the settings.
- **default():** returns the settings to default.

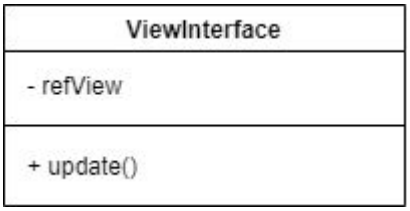
3.1.4 Game



Attributes:

- **GameType[]:** keeps the type of game.

3.1.5 ViewInterface



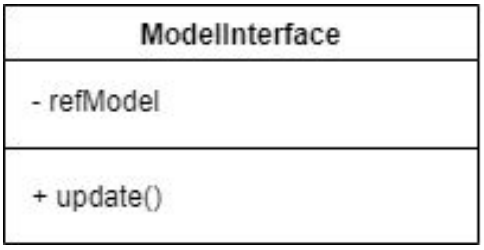
Attributes:

- **refView:** reference to a target view.

Methods:

- **update():** updates the current view.

3.1.6 ModelInterface



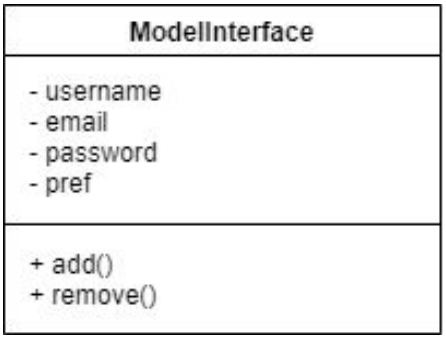
Attributes:

- **refModel:** reference to a target model.

Methods:

- **update():** updates the model.

3.1.7 UserData



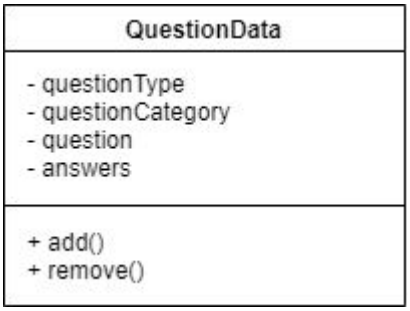
Attributes:

- **username:** keeps the username of the user.
- **email:** keeps the email of the user.
- **password:** keeps the password of the user.
- **pref:** keeps the settings preferences of the user.

Methods:

- **add():** adds a new user.
- **remove():** removes a user.

3.1.8 QuestionData



Attributes:

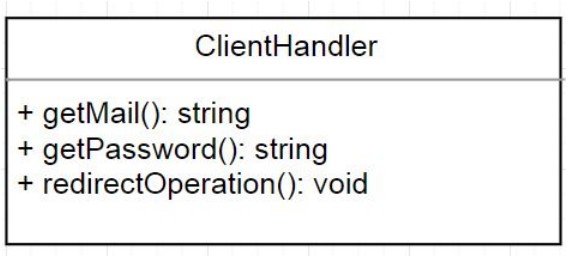
- **questionType:** keeps the type of the question.
- **questionCategory:** keeps the category of the question.
- **question:** keeps the question itself.
- **answers:** keeps the multi-choice answers of the question.

Methods:

- **add():** adds a new question.
- **remove():** removes a question.

3.2 Logic Tier Interfaces

3.2.1 ClientHandler



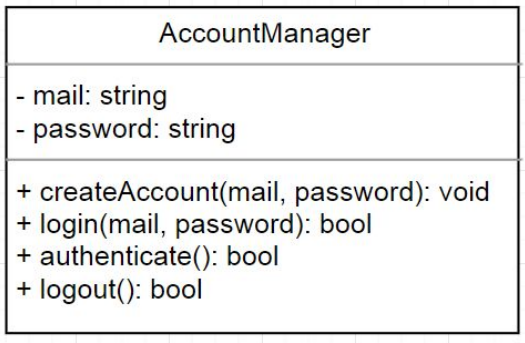
Methods:

● **getMail():** get the e-mail address of the user.

● **getPassword():** get the password of the user.

● **redirectOperation():** determines which operation will be performed based on the input from the user.

3.2.2 AccountManager



Attributes:

● **mail<string>:** represents the e-mail address of the user.

● **password<string>:** represents the password of the user.

Methods:

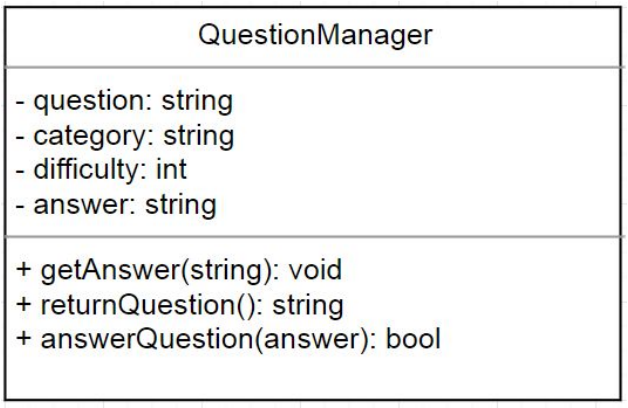
● **createAccount(mail, password):** creates new account for the user.

● **login(mail, password):** logs in by using the given e-mail address and password.

● **authenticate():** authenticates user account.

● **logout():** the user logs out.

3.2.3 QuestionManager



Attributes:

● **question<string>:** the content of the question that is being handled.

● **category<string>:** the category of the question that is being handled.

● **difficulty<int>:** the difficulty of the question that is being handled.

● **answer<string>:** the answer that is taken by the user.

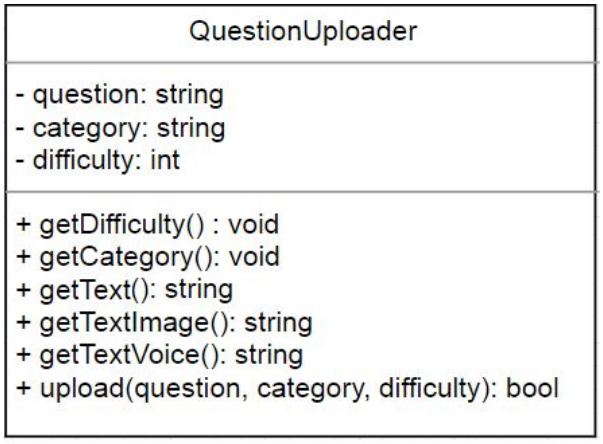
Methods:

● **getAnswer():** get the answer from the user.

● **returnQuestion():** the content of the question is returned to be shown to the user.

● **answerQuestion(answer):** whether the given answer is the correct answer is being checked.

3.3.4 QuestionUploader



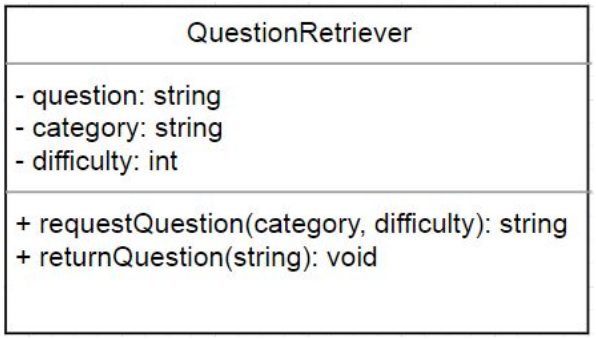
Attributes:

- **question<string>:** the content of the question to be uploaded.
- **category<string>:** the category of the question to be uploaded.
- **difficulty<int>:** the difficulty of the question to be uploaded.

Methods:

- **getDifficulty():** gets the difficulty of the question.
- **getCategory():** gets the category of the question.
- **getText():** get the content of the question.
- **getTextImage():** getS the content of the question to be uploaded with an image using API.
- **getTextVoice():** getS the content of the question to be uploaded with speech.
- **upload(question, category, difficulty):** uploads the question by with the given content, category and difficulty by communicating with the data tier.

3.3.5 QuestionRetriever



Attributes:

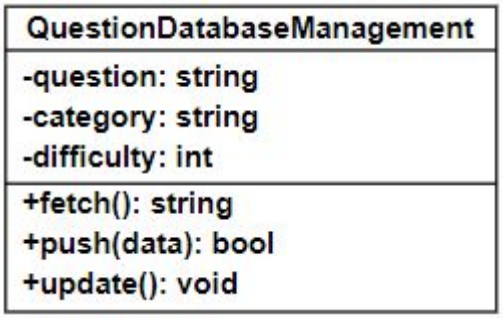
- **question<string>:** the content of the question to be retrieved.
- **category<string>:** the category of the question to be retrieved.
- **difficulty<int>:** the difficulty of the question to be retrieved.

Methods:

- **requestQuestion(category, difficulty):** requests a question from the data tier with the given category and difficulty.
- **returnQuestion(string):** returns the content of the question to be shown to the user.

3.3 Data Tier Interfaces

3.3.1 Question Database Management



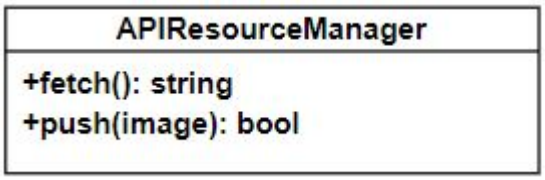
Attributes:

- **question<string>:** represents the question itself.
- **category<string>:** represents the category of the question.
- **difficulty<int>:** represents the difficulty level of the question.

Methods:

- **fetch():** This function fetches question from the question database.
- **push(data):** This function uploads a question to the question database.
- **update():** This function updates a question in the question database.

3.3.2 API Resource Management

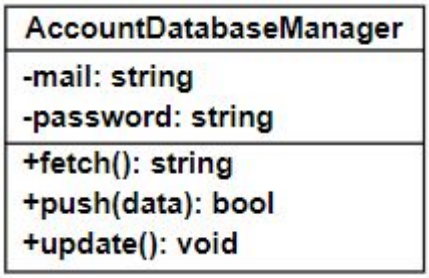


Methods:

● **fetch():** This function fetches the string that is scanned from an image from the API resource database.

● **push(image):** This function pushes an image to be scanned to the API resource database.

3.3.3 Account Database Management



Attributes:

● **mail<string>:** represents the mail address of the user.

● **password<string>:** represents the password of the user.

Methods:

● **fetch():** This function fetches user data from the account database.

● **push(data):** This function pushes user data to the account database.

● **update():** This function updates user data in the account database.

4. References

[1] Dünyada 284 milyon görme engelli var https://www.haberturk.com/saglik/haber/997825-dunyada-284-milyon-gorme-engelli-var. [Accessed: 14-Oct-2019].

[2] Türkiye'de 220 bin görme engelli var http://www.kuzeyhaber.com/haber/guncel/turkiyede-220-bin-gorme-engelli-var/71622 . [Accessed: 14-Oct-2019].

[3]Text To Speech Reader https://ttsreader.com. [Accessed: 9-Oct-2019].

[4] Voice Control:Everything you need to know https://www.imore.com/voice-control-everything-you-need-know. [Accessed: 9-Oct-2019].

[5] Android'i TalkBack ile kullanmaya başlama https://support.google.com/accessibility/android/answer/6283677?hl=tr. [Accessed: 9-Oct-2019].