University of Stavanger

Faculty of Science and Technology

MASTER'S THESIS

Study program/ Specialization:	Spring semester, 20...... Open / Restricted access
Writer:	………………………………………… (Writer's signature)
Faculty supervisor: External supervisor(s):	
Thesis title:	
Credits (ECTS):	
Key words:	Pages: ………………… + enclosure: ………… Stavanger, ……………….. Date/year

University of Stavanger

Faculty of Science and Technology
Department of Electrical Engineering and Computer Science

Mobile ios/Android app for fake news detection game

Bachelor's Thesis in Computer Science
by

Minh Christian Tran

Karam Tamim Yanis

Internal Supervisors

Vinay Jayarama Setty

Supervisor 2

External Supervisors

External Supervisor 1

External Supervisor 2

Reviewers

Reviewer1

Reviewer2

May 14, 2021

"A delayed game is eventually good, but a rushed game is forever bad."

Shigeru Miyamoto

Abstract

The surge of fake news have risen drastically with the increasing popularity of social media and forums as of late causing major threats to various agencies and sectors, and as more fake news starts to take over mainstream media, readers have also become less critical to what they are reading. This problem has led people to blindly believe everything they read and in some cases even create their own conspiracy theories about certain topics. These actions often leads to major misunderstandings and could cause major damage to everyone if it gets to extreme. In this work we propose a mobile application that educates the users about detecting fake news in a form of a game. Its purpose is to make the users more critical to what they are reading as well as educate the users about current hot topics that flows around in mainstream media.

The point of this mobile application is to create a game where the player has to guess if the claim is true or false based on the evidence given to the player. Less time needed to guess the correct answer results in a higher score.

Singleplayer game: A user plays as the guesser while the computer acts as the proposer that selects hints to show the guesser. At the singleplayer game, you have to ask for the hints yourself. The proposer uses an algorithm that selects hints at random.

Multiplayer game: A user can either create a new game or join an existing game with one or several players depending on how big the creator of the lobby chose to make the lobby. If a new game is created, the user is given the role as proposer and have to wait until the lobby is full before the game can start.

When the game starts, the proposer is presented with a claim and a hint given by the proposer. The guesser then have to guess whether the claim is true or false. If the answer is wrong, the user receive zero points. If the guesser makes a correct answer, the guesser is rewarded with points that vary on how fast the guesser managed to guess the correct answer. The proposer gets their score calculated by the average score of the guessers.

Acknowledgements

We would like to thank our supervisor Vinay Jayarama Setty at the Department of Electrical Engineering and Computer Science at University of Stavanger. Creating our first ever mobile application without any prior experience seemed really difficult. Thankfully did his guidance and expertise help us throughout the thesis.

Contents

Abbreviations

UI	**U**ser **I**nterface
OS	**O**perating **S**ystem
API	**A**pplication **P**rogramming **I**nteface
SDK	**S**oftware **D**evelopment **K**it
REST	**RE**presentational **S**tate **T**ransfer
JSON	**L**ist **A**bbreviations **H**ere
CPU	**C**entral **P**rocessing **U**nit
PHP	**H**ypertext **P**reprocessor
SQL	**S**tructured **Q**uery **L**anguage
ORM	**O**bject **R**elational **M**apper
URL	**U**niform **R**esource **L**ocator
HTTP	**H**HyperT**ext **T**ransfer **P**Protocol
HTTPS	**H**HyperT**ext **T**ransfer **P**Protocol **S**ecure
SSL	**S**ecure **S**ockets **L**ayer

Chapter 1

Introduction

1.1 Motivation

In the information era we live in today, where information is crucial among people, being fed misinformation can cause dangerous situations in our society. As new information and stories keep getting published at a faster rate without verification, figuring out what is true and what is false becomes increasingly harder to do. Fake news has gone from being sent via emails to practically being everywhere on the Internet, especially in social medias. In the 21th century, the ability to create fake news has never been easier than before thanks to the popularity of social media. People are able to generate financial profits through social media by deceiving readers into clicking links, thus maximizing traffic and profit. Creating fake news has even become a job for some as generating profit and spreading those fake news has become so easy[1].
According to Dictionary.com, the definition to fake news is: "false news stories, often of a sensational nature, created to be widely shared or distributed for the purpose of generating revenue, or promoting or discrediting a public figure, political movement, company, etc"[2]. Social medias such as Facebook, twitter, Instagram and blogs have recently become popular news providers, and along with the rise of news on social media, fake news become more prevalent. Teens, especially, are vulnerable to these fake news as they tend to use social media to read news and observe what is happening in the world. Fake news has become frighteningly leaked in many countries of world to spread an atmosphere of panic and fear of disease or to mix facts about the Corona pandemic and Corona vaccine, as the period of crisis is the best environment for the emergence of fake news. As the pandemic is ongoing, having fake news about the pandemic being spread around is the last thing we need as it will only apply gasoline to the fire. If fake news is able to carry as much power as it has on a daily basis, it is only a matter of time before

society crumbles. People, especially teenagers and future generations should be able to distinguish the fake news they read on the internet from real ones. This mobile game should help educate people in a fun and interesting way. Thus, the goal is to create a mobile game that can educate its users about what trending news is fake and what is not.

1.2 Goal

There are websites such as `https://www.politifact.com/` who does manual fact checking on mainstream topics and then publish articles that explain why they are fake news or not. These websites does fact checking by manually gathering evidence and sources as to whether claims and statements flowing around in the internet is true or not. These articles can sometimes be a bit tiring and boring to read and are not so appealing to younger audiences.
The first goal of the present thesis is to create a fun way to read and learn about fake news. Instead of just reading about the news, the user can instead try out their own knowledge about the world news and test out their ability to detect fake news. The user will become more educated about subjects flying around in the internet while still have fun doing so. The game can be played both alone and with friends, making it a fun party game where you can test your knowledge against your friends as well as an educational activity.
The second goal is to collect data about what the average users think is important in detecting fake news as well as how informed the users are regarding fake news flowing around the internet.

1.3 Outline

The thesis is outlined as follows:
Chapter 2 introduces the background theory about the thesis. The focus is especially on tools used in the thesis.
Chapter 3 describes the different components of the server.
Chapter 4 describes the backend of the application.
Chapter 5 describes the frontend of the application.
Chapter 6 is the experimental evaluation
Chapter 7 is the discussion part of the thesis.

Link to the github of the project:

Frontend: `https://github.com/Mintra99/BachelorOppgave`

Backend: `https://github.com/karamyanes/fakenews-app.com`

API scraper: `https://github.com/karamyanes/politifact-scraper`

Website: `https://fakenews-app.com/admin/`

username: karam, password: karam

Chapter 2

Theoretical Background

2.1 Introduction

This chapters goal is to present the background material related to this project as well as an introduction to the frameworks used to create the application. The development of the application will be explained more in depth later in chapter 4 and 5.

2.2 Related Work

There are some related work in terms of fake news games. Bad News, Fake It to Make It and Factitious are games that was created to train and sensitize readers to detect and stay immune to fake news[3–5]. Bad News and Fake It to Make It are games that gives the players a firsthand experience about what goes into creating fake news and how it is spread. Factitious is a game more similar to the mobile application we are creating in that the game gives a real news article the player then have to decide whether the article is true or not. Researchers and journalists who with their knowledge and experience, have designed these games to make people more conscious of fake news distribution and how to detect them.

2.3 The history of Mobil App and fake news

If you go back to the history of mobile apps, you can clearly know that some Java games, calculator or monthly calendar were all under the mobile app category.
These things weren't called "applications" at the time, and are generally referred to as "features" in the "Mobile Office" section of the phone, for example this alarm clock

(app). Earlier it was some Java games, calculator, or calendar is all under the mobile app category, and Nokia is still remembered for the popular Snake game on some of its older phones.
Then Apple App Store launched with 500 apps, meaning there is no "true" first app. Then Google followed Apple and launched Google Play.[6]
In 2017, 'fake news' became Collins Dictionary's word of the year and it's remained in the headlines ever since. Although the phrase might appear to be a modern invention, examples of it can be found throughout history.
Propaganda has been used in wars throughout history to try and change people's views. This is a type of fake news where false information is used for political gain. It can help change public opinion by persuading people that their country should go to war, or convince them the other side is their enemy.
In recent years, there has been an explosion of fake news as false stories are shared widely on social media without being fact-checked. Cheap and portable access to the internet across the globe means stories can spread in a matter of seconds and minutes rather than days or weeks. Many of these stories are completely made up and can make money from advertising - the more clicks a website gets, the more money it makes.[7]
In Belgium Police clash with April/2021 Fool's Day party crowd.dispersed some 2,000 people gathered in the "Bois de La Cambre" park in Brussels in defiance of Belgium's coronavirus lockdown.Thousands showed up at the fake concert, which promised a performance by producer Calvin Harris and a one-off reunion of French band Daft Punk.The party was announced on Facebook.[8]
The future of fake news is more dangerous because videos were usually a fairly reliable source of news, as they can't be faked as easily as a photograph or headline. However, the reliability of video sources is now under threat from deepfakes. These are videos that use computer software and machine learning to create a digital version of someone. Normally it's used to put the face of a celebrity or politician on to someone else's body.[7]

2.4 Game theory

Game theory is a study of mathematical models of how the human brain solves problems. Game theory have applications in fields like social science, logic, system science and computer science[9]. Game theory was given its first general mathematical formulation by John Von Neuman and Oskar Morgenstern in 1944 with the intention of understanding strategy[10]. The main focus of game theory is to serve as a model of interactive situation among rational players. Interactive decision-making, where the outcome depends on the action of each participant is key to this theory.

In any situation with more than two participants involved with known payouts or consequences, game theory can be applied, whether it is a real life situation or not. Today, most businesses and economics use Game theory as a way to predict future outcomes and to maximise their profits based on their predicted outcomes. Still, there are limitations to game theory. The issue with game theory is that it relies on the assumption that people are rational thinkers with self-interest in mind[11]. Game theory can therefore not account for situation where people decides not to cooperate with common logic and do irrational decisions.

2.5 Flutter

Flutter is an open-source UI , software development kit created by Google. It is made for the purpose of developing applications for OS like Android, iOS, Linux and Windows to name a few. Flutter apps are written in their own language called Dart. Flutter also have an engine which is primarily written in C++ and provides low-level rendering support using Skia graphics library. Flutter was used to create the application as well as designing the front-end of everything in the application.
Flutter consists of two important parts:
1- An SDK : A collection of tools that are going to help you develop your applications. This includes tools to compile your code into native machine code (code for iOS and Android).
2- A Framework UI: A collection of reusable UI elements (buttons, text inputs, sliders, and so on) that you can personalize for your own needs.
To develop with Flutter, you will use a programming language called Dart. The language was created by Google in October 2011, but it has improved a lot over these past years. Dart is a client-optimized language for developing fast apps on any platform. Its goal is to offer the most productive programming language for multi-platform development, paired with a flexible execution runtime platform for app frameworks. Dart provides the Flutter and runtimes that power Flutter apps, but Dart also supports many core developer tasks like formatting, analyzing, and testing code.

2.6 Django

It is a free and open source web framework. Web frameworks provide tools and libraries to simplify common web development operation, after that we need to add our piece of code , so the problem will be solved and this make programming much easier.

Django written in the Python programming language, established in 2005, and its main goals are to facilitate the process of developing complex web sites, in addition to that it adopts the MVT-MVC architecture system to build projects (models that deal with data, presentations and determine the data that will be presented, as for Templates determine how the display appears on the browser). Main advantages of Django:
1- Fast: it won't take long to code your ideas. Django is an ideal solution for developers who focus on productivity and get their work done on time without a hassle.
2- Tons of packages: Django includes dozens of extra you can use to handle common web development tasks.
3- Security: Django is strong in this respect as well, as it creates excellent solutions to many potential problems, such as SQL injection, fraud vulnerabilities and more.
4- Scalability: Django provides innovative solutions to congestion problems.
5- Versatile: Django is used to build all sort of things from content management systems to social networks to scientific computing platforms.
There are several other web frameworks that compete with Django now,including those written in Python as the Flask framework, and others in other languages, the most important of which are: Laravel written in PHP, and the Ruby on Rails framework for the programming language.

2.7 Swagger

Swagger UI allows anyone to be it your development team or your end consumers to visualize and interact with the GlsAPI's resources without having any of the implementation logic in place. It's automatically generated from your Open API (formerly known as Swagger) Specification, with the visual documentation making it easy for back end implementation and client side consumption.[12]
Swagger automatically captured the details from the GlsAPI, it reads the various methods presents in the API and put together in a flexible UI screen and if there is any change in the REST API it automatically read the updated code and update the swagger UI just by restarting the application.
Benefits of Swagger:
In addition to its goal of standardizing and simplifying API practices, a few additional benefits of Swagger are:
1- It has a friendly user interface that maps out the blueprint for GlsAPIs.Documentation is comprehensible for both developers and non-developers like clients or project managers.
2- Specifications are human and machine readable.
3- Generates interactive, easily testable documentation.

4- Supports the creation of API libraries in over 40 languages.

5- Format is acceptable in JSON and YAML to enable easier edits.

6- Helps automate API-related processes.[13]

Chapter 3

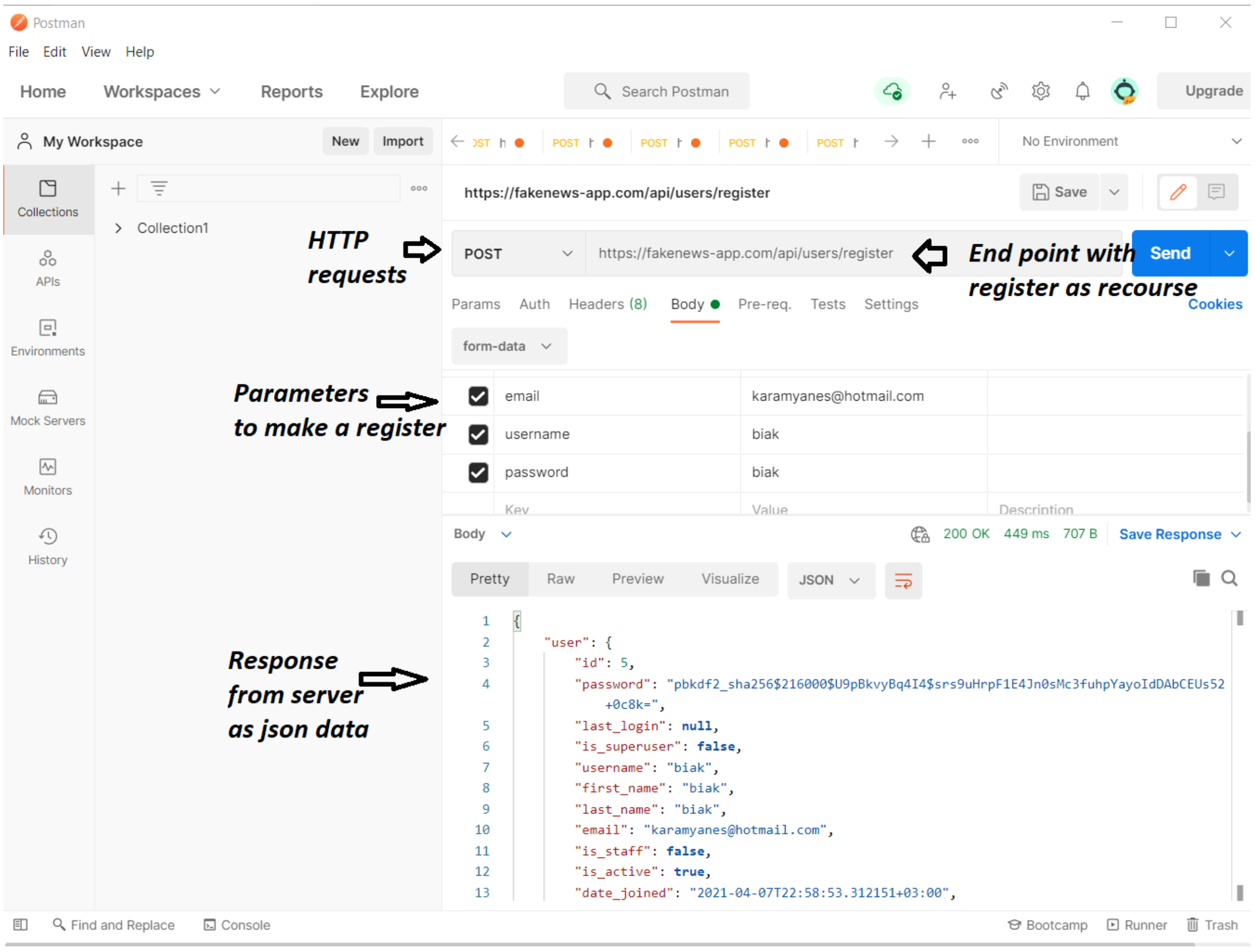
Server

3.1 Introduction

The server served as an API for the mobile application. Requested data such as claims, sources and hints from the application is gathered by the server and ready to be displayed to the user when starting a game of fake news detection. This chapter explains how the server side and API of the application operates.

3.2 API

In a client-server communication, REST suggests to create an object of the data requested by the client and send the values of the object in response to the user. It is simple and standardized approach of communication, it is scalable and stateless and it has high performance and support caching.
Postman is a collaboration platform for API development. Postman's features simplify each step of building an API and streamline collaboration so you can create better APIs—faster.[14]This is done by allowing users to create and save simple and complex HTTP/s requests. It has the ability to make various types of HTTP requests(GET, POST, PUT, PATCH), as well as read their responses. The result - more efficient and less tedious work. Converting the API to code for various languages(like JavaScript, Python).
As well, Swagger automatically captured the details from the API.



https://fakenews-app.com/api/users/register

POST https://fakenews-app.com/api/users/register

✓	email	karamyanes@hotmail.com
✓	username	biak
✓	password	biak

200 OK 449 ms 707 B

```
{
    "user": {
        "id": 5,
        "password": "pbkdf2_sha256$216000$U9pBkvyBq4I4$srs9uHrpF1E4Jn0sMc3fuhpYayoIdDAbCEUs52
            +0c8k=",
        "last_login": null,
        "is_superuser": false,
        "username": "biak",
        "first_name": "biak",
        "last_name": "biak",
        "email": "karamyanes@hotmail.com",
        "is_staff": false,
        "is_active": true,
        "date_joined": "2021-04-07T22:58:53.312151+03:00",
```

Figure 3.1: Request and Response by using Postman

3.2.1 Django Rest Api

- HTTP requests will be matched by Url Patterns and passed to the Views.
- Views processes the HTTP requests and returns HTTP responses (with the help of Serializer).
- Serializer serializes/deserializes data model objects.
- Models contains essential fields and behaviors for CRUD Operations with Database.[15]

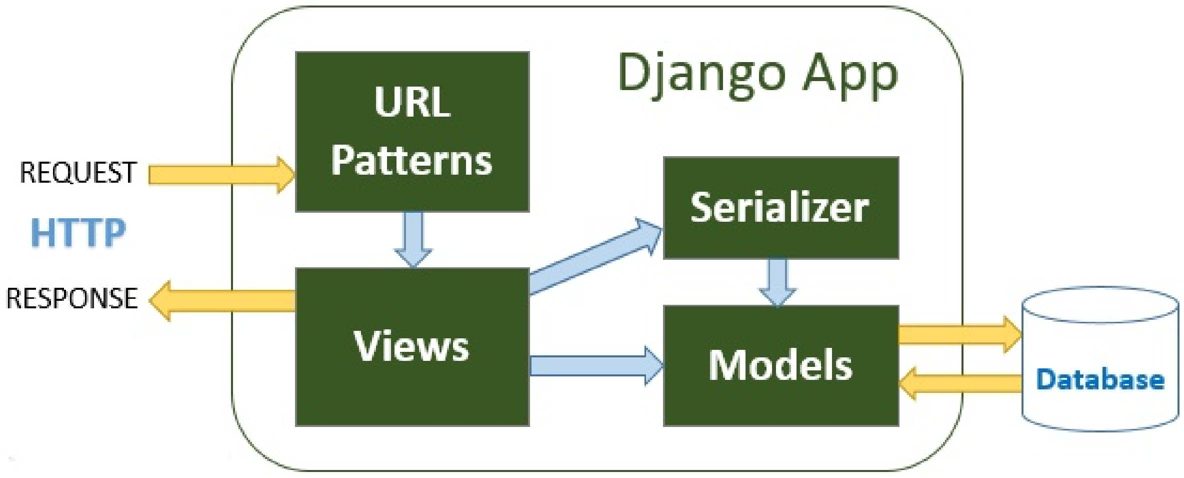


Figure 3.2: Django Rest Api Architecture

3.2.2 Request and Response

Request start with the end point for example https://fakenews-app.com/api/users/register, "register" is known as recourse and recourse is important for REST API, Request is send from the client to the server and the Respond is received back from the server.
CRUD stands for Create, Read, Update, and Delete. But put more simply, in regards to its use in RESTful APIs, CRUD is the standardized use of HTTP Action Verbs. This means that if you want to create a new record you should be using "POST." If you are trying to read a record, you should be using "GET." To update a record utilizing "PUT" or "PATCH." And to delete a record, using "DELETE".[16]
Request has an operation (GET,POST,PUT,DELETE), and should has end point (http://localhost:3000/api/users/register), and might also has parameters or body ("username": "karam","password": "karam", "email": "karam@hotmail.com"), and finally header this is a special part of API Request which might have things like an API key or some Authentication data.
Response will come in form of JSON data.

3.2.3 POST

Whenever create a form that alters data server-side, use method="post". The user have the option to create a lobby when playing multiplayer. When creating the lobby, the database receive several POST requests from the applications frontend. The first POST request the database receive from the API when creating the lobby is a POST request

containing the lobby name, number of players and a list of the claims that are going to be used in the lobby. This POST request is done by two functions called "createGame" and "addGameQuestions".
The createGame function is responsible for creating the lobby and sends to the database lobby name and lobby size.The endpoint for POST method: $serverUrl/game/Endpoint Name/. Endpoint need a body to enter the data required to determine the name of the game and the number of players.
Where:

- $serverUrl:"https://fakenews-app.com/api".
- Endpoint Name : game_name.

Body:

- game_name: $game_name that the creator of the game choose it.
- num_of_players: $number the creator of the game decide how many player will play.

```
createGame(String game_name, String numPlayers, List listOfClaims) async {
    String myUrl = "$serverUrl/game/new_game/";
    final response = await http.post(myUrl, headers: {
      'Authorization': 'Bearer $value'
    }, body: {
      "game_name": "$game_name",
      "num_of_players": '$number',})}
```

The addGameQuestions function is responsible for adding the claims to said lobby and sends the database a list of the claims. This data that the database receive is then used in the backend to create the lobby and add the creator of the game to the lobby.The endpoint for POST method: $serverUrl/game/Endpoint Name/. Endpoint need a body to enter the data required to determine the id to the game that the list of claims will be add to and the id of claims that the list has.
Where:

- $serverUrl:"https://fakenews-app.com/api".
- Endpoint Name : lobby_question.

Body:

- game_id: $gameId the id of the game that the creator has made it.

- question_id: $question_id the id of questions that add to the game

```
addGameQuestions(List questions) async {
        String myUrl = "$serverUrl/game/lobby_question/";
         final response = await http.post(myUrl, headers: {
        'Authorization': 'Bearer $value'
      }, body: {
        "game_id": "$gameId",
        "question_id": "$question_id",}))}
```

When joining a lobby in multiplayer, there is also one POST request done to the database. The function joinGame is called when a player joins a lobby. This function performs a POST request to the database, giving the database the data about which lobby the player wants to join for the backend to later add the player to said lobby. Whenever the user answer correctly on a claim in either singleplayer or multiplayer, a POST request is sent sending the users updated score to the database. In multiplayer, the updated score by the guesser is then used by the proposer to update their own score through the same POST request.

3.2.4 GET

Playing either the singleplayer or multiplayer game mode triggers a GET request from the frontend of the application to the server asking for the hint that are stored up in the database. The endpoint for GET method: $serverUrl/game/EndpointName/$gameId/$claimId/
Where:

- $serverUrl:"https://fakenews-app.com/api".
- Endpoint Name : get_hint.
- gameId: is the current game number.
- claimID: is the current claim number.

```
getHint(int claimId) async {
   String myUrl = "$serverUrl/game/get_hint/$gameId/$claimId/";
   if (claimId != null) {
      final response = await http.get(myUrl, headers: {
        'Authorization': 'Bearer $value'})}
```

Algorithm 3.1 GET request for claims

```
mapResponse: claim, hint, source   _usernameController ← TextEditingController()
_passController ← TextEditingController()
procedure FITCHDATA(URL)
    mapResponse ← responseBody
claim, hint, source
```

The API response contained article metadata, claims, sources and hints.

When playing as the proposer in the multiplayer game mode, a GET request is triggered at the end of each claim. This GET request gathers the points of the guessers in the lobby from the database. The gathered score is then used to give the proposer its own score based on the collective performance of the guessers.

Algorithm 3.2 GET request for score

```
mapResponse: claim, hint, source  userStatus ← String status  userID ← int id
procedure GETSCORE(userStatus, userID)
    mapResponse ← responseBody
score
```

.
..
...
....

3.2.5 PUT

Put is used for updating an object. This is an example used the PUT "HTTP method". For sending claim's hint, in-case the 1st player (proposer) wants to send a hint for the other players (guessers) , so they can choose the correct answer for the claim.

Scenario:

- proposer will create a new game ..
- claims will be add by default to start the game and both players will see the same claims.
- proposer will see a button to send hint to guesser
- Send hint button is a "PUT Method"

- Calls the API with ‘QuestionID‘, ‘GameID‘ and ‘hint‘.
- Method will append the text sent in "Hint" to the API and it will be saved in db, then guesser will be able to see hint if requested.

The "PUT" method has a body like post method. Also in our case we are keeping our endpoints restful as much as possible. The endpoint for the send hint "PUT" method: $serverUrl/game/EndpointName/gameId/claimID/.Endpoint need a body to enter the data required.

Where:

- $serverUrl:"https://fakenews-app.com/api".
- EndpointName: lobbyDoc.
- gameId: is the current game number.
- claimID: is the current claim number.

body:

- "doc_hint": $docHint to enter the text and saved in db.

```
addGameClaim(int claimId, String docHint) async {
        if (claimId != null) {
      final response = await http.put(myUrl, headers: {
        'Authorization': 'Bearer $value'
      }, body: {
        // "game_id": "$gameId",
        // "question_id": "$claimId",
        "doc_hint": "$docHint",
      })}
```

3.3 Docker

Docker is a tool designed to make it easier to create, deploy, and run applications by using containers. Containers allow a developer to package up an application with all of the parts it needs, such as libraries and other dependencies, and deploy it as one package.

Docker is completely compatible with any operating system and most of the programming languages environments. Docker container can deploy to any machine without any compatibility issue, these containers running in computer or server and act like little

micro computers with very specific jobs, each with their own operating system and their own isolated CPU, memory and network resources, and because of this they can be easily added, removed, stopped and started again without effecting the host machine.
Dockerfile is simple text document that instructs the Docker image that will be built, image start with word 'FROM' which can find a container to use from the Dockerhub and command 'RUN' to do downloading, installing and running software.

In our case we used containers for:

- Api Python container, which is used for the API REST Methods.
- Database MySQL container, which is used for saving data in MySQL DB using python API in a local network between API Container MySQL BD container.
- Adminer container, which is used for managing MySQL database.

3.4 website (Host and Domain)

Android studio does not accept API which has local hostURL, so it is necessary to buy host and domain. Domain Name is website address that people type in the browser bar to visit site. In other words, if we assume that your website is a home, then your domain will be its address.A domain name can contain words that make it easy to remember website addresses, IP address is a combination of numbers separated by point. Usually, the IP addresses look like this: 66.249.66.1 people cannot remember these numbers and use them to log onto websites.
Domain names and web hosting are two different services. However, they do work together.Without domain names, it will be difficult for people to find a website and without web hosting, no one can create a website in the first place. When someone enters site's domain into the browser, the domain (domain name) is converted to the IP address of the web hosting company's computer. This computer contains the files for website, and sends those files back to the users' browsers.
To let the frontend call the backend by using API, it need Webhosting and since Android studio connect only with website with a webserver, webhosting is necessary to make the connection.
Game application is hosted on a Linux servers and on domain name `https://fakenews-app.com/admin/`.
Application is working as the following:

- Domain: http://fakenews-app.com/admin/

- PuTTY is a tool , to access server.
- Host: Linux server connected to domain with name servers.
- SSL certificate installed on Linux server.
- Dockers, containers includes our Python API application.

PuTTY is a software terminal emulator for Windows and Linux. It provides a text user interface to remote computers running any of its supported protocols, including SSH and Telnet.[17] This tool is used to access, manage and deploy :

- API and MySQL containers .
- Git and git branches from remote github repo.
- SSL certificate and renewing it

Chapter 4

Backend Mobile Application

4.1 Data Scraping

4.1.1 Web Scraper

Data scraping, also known as web scraping, is the process of importing information from a website into a spreadsheet or local file saved on your computer[17]. It's one of the most efficient ways to get data from the web.
In the absence of API for the web that we want to get data from, web scraper is very important tool for data scientist because the entire internet becomes in form of database. Since`https://www.politifact.com/` has no API to picking up some data, web scraper will be a great idea to bring the data from `https://www.politifact.com/`.
Python Beautiful Soup (Bs4) `https://pypi.org/project/beautifulsoup4/`) library is specialized for pulling data out of HTML and XML files. Python Beautiful Soup library makes it easy to scrape information from web pages. It sits atop an HTML or XML parser, providing Pythonic idioms for iterating, navigating, searching, and modifying the parse tree. It works with your favorite parser. It commonly saves programmers hours or days of work.[18]

-

Web Scraper Usage

:

As previously explained that *politifact.com* does not have an API, the *Python Beautiful Soup (Bs4)* output needed for adding the recent and scrapped data from *politifact.com* to the Game. Saving *Beautiful Soup* output in *JSON format.* Which was perfect solution in current case, *JSON format* makes the importing data in Python API more easier.

-

Data Cleaner / Python Tokenize

:
As mentioned previously that *Beautiful Soup* output in *JSON format*, So it was very important to do clean the data before processing it to the format needed in the Python "data importing process".
Cleaner.py is a simple algorithm that cleans the data generated by *Bs4* then applying a function to tokenize the document hints that will be sent from the Proposer (player1) to the Guesser (player2).

Here is a sample of cleaner.py script:

```
import json
import nltk
import time
timestr = time.strftime("%Y%m%d-%H%M%S")
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
def tokenize(doc_text):
sentences = nltk.sent_tokenize(doc_text)
    return sentences
with open("politifact_claims.json", "r") as jsonFile:
    data = json.load(jsonFile)
    for d in data:
        new_doc = tokenize(d['fields']['doc'])
        if new_doc:
            d['fields']['doc'] = new_doc
        else:
            d['fields']['doc'] = 'empty hint'
jsonFile = open("politifact_%s.json" % timestr, "w+")  # generating file with timestamp
jsonFile.write(json.dumps(data))
jsonFile.close()
```

Now .. Data is ready to be imported synced in Python API database.

4.2 Database

SQL is the language we use to issue command to database

- Create / Insert data
- Read / Select some data
- Update data
- Delete data

Django use ORM which allow us to to map tables to objects and columns, this objects used to store and retrieve data from the database and commands and run those ORM improve probability across database dialects (SQlite, MySQL, Postgres, Oracel)

4.2.1 Models

For Django API project, in each application there is a model.py file.
In current case: there is game/models.py note that game application folder.
Main and First class called Lobby and refers to the name of table in SQL with extend or inherent (models.model) which give a lot of fixtures and functionalities, and then create a couple of fields that basically says character field models is needed or integer field models is needed or other kind of models could be needed.
This example model define a Lobby which has game_name, num_of_player , current_players and created_at:

```
from django.db import models

class Lobby(models.Model):
    game_name = models.CharField(max_length=20)
    num_of_players = models.IntegerField(default=2)
    current_players = models.IntegerField(default=0)
    created_at = models.DateTimeField(auto_now_add=False, null=True)

python manage.py makemigrations'
python manage.py migrate'
```

game_name, num_of_player, current_players and created_at are fields of the model.
Each field is specified as a class attribute, and each attribute maps to a database column.
No need to care about primary key because Django by default gives each model an auto-increment primary key.
In class answer choices are a sequence of 2-tuples to use as choices for this field. If this is given, the default form widget will be a select box instead of the standard text field and will limit choices to the choices given

```
class Answer(models.Model):
    STATUS=(
        ('true','true'),
        ('barely true','barely true'),
        ('false','false'),
        ('mostly true','mostly true'),
        ('pants on fire','pants on fire'),
        ('half-true','half-true'),
    )
    answer_text = models.CharField(max_length=256,choices=STATUS)
    questionid = models.ForeignKey(Question, on_delete=models.CASCADE,
    verbose_name = "related to Question")
    is_correct = models.BooleanField(default=False)
```

The first element in each tuple is the value that will be stored in the database. The second element is displayed by the field's form widget.[19]
where models is class, and CharField is a method within that class and the next step is to build by called:

- python manage.py run migration then Django will make this table.
  manage.py is like the file that start up every thing migration a set of migration scripts and see that is making these files and their actual files like 0001 initial.py, and to continue building by called:
- python manage.py run migrate which is reading the migration and changing the database.

They are kind of Django internal files that no need to write them by the programmer, but the migrate will read the 0001 initial.py file and actually generated SQL commands and run those SQL commands for the developer.

Django REST Framework DRF which is a powerful and flexible toolkit built for Django framework and it used for building and developing restful Web API. Rest Framework help to get the data from database and output it into format can be read by other technology like mobile application, so basically this happen when we convert data to json by help from serializers and postman helps to work test these data.

There are some reasons to use Django REST_Framework:

1. It has a web browsable API is a huge usability win for developers.

2. It has authentication policies

3. It has Serialization that supports both ORM and non-ORM data sources.

4. It has a good documentation and great community support.

Serializer is very important to send data to user which first convert the data to json. Serializer also provide deserialization, allowing parsed data to be converted back into complex types.
The first thing creating migration and migrate is to provide a way of serializing and deserializing the model into representations such as json.There is two main types of serializer:

- General serializers (serializers class).
- ModelSerializers.

ModelSerializers is less customise than serializers class, but more simple to use.
fields = ' ___all___ ' to have a serialzer representing all the fields of Lobby model.

```
from .models import Lobby, Player, Question, Answer, LobbyQuestion
from rest_framework import serializers


class LobbySerializer(serializers.ModelSerializer):
        class Meta:
                model = Lobby
                fields = "__all__"
```

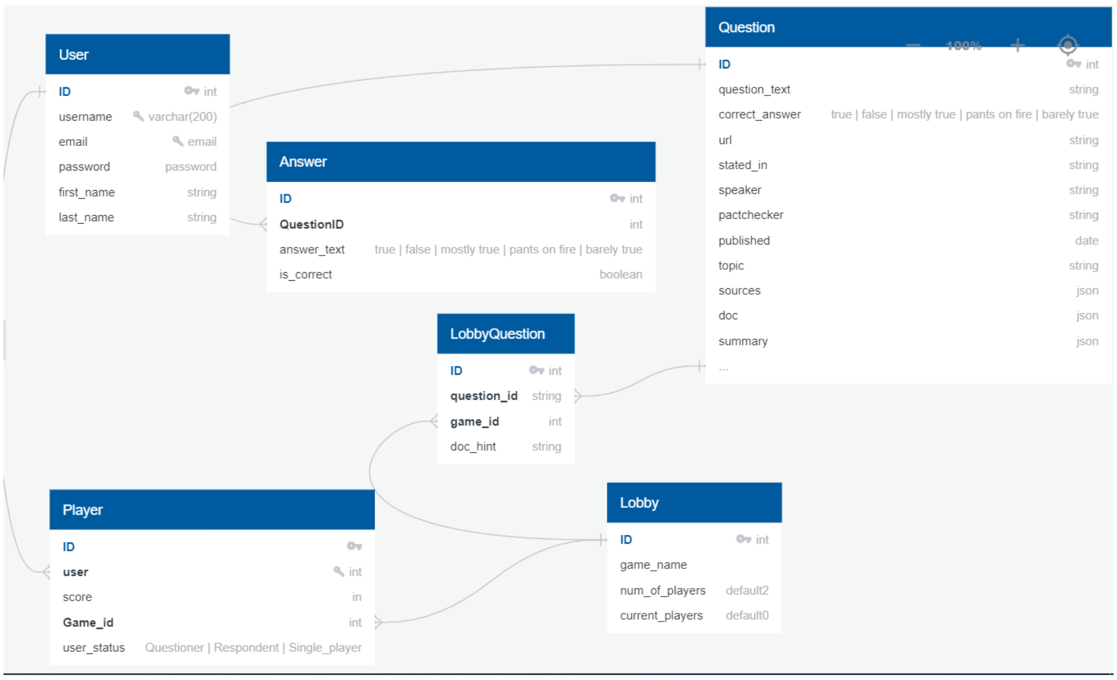


Figure 4.1: Database

4.2.2 QuerySets

Once data models has been created, Django automatically gives a database-abstraction API that lets you create, retrieve, update and delete objects.[20]

Create or Post

To create an object, instantiate it using keyword arguments to the model class. At the first Permission is needed to be sure that only permit user can make Post, Get or Put.

```
permission_classes = [permissions.IsAuthenticated]
```

Call is_valid during deserialization process before write data to DB. is_valid perform validation of input data and confirm that this data contain all required fields and all fields have correct types. raise_exception=True, this exception and return 400 response with the provided errors in form of list or dictionary

```
def post(self, request, *args, **kwargs):
                serializer = self.get_serializer(data=request.data)
                serializer.is_valid(raise_exception=True)
```

Getting the parameter sent in the POST method , and assigning it into question_id object, getting the parameter sent in the POST method , and assigning it into game_id object

```
question_id = request.POST['questionid']
game_id = request.POST['game_id']
```

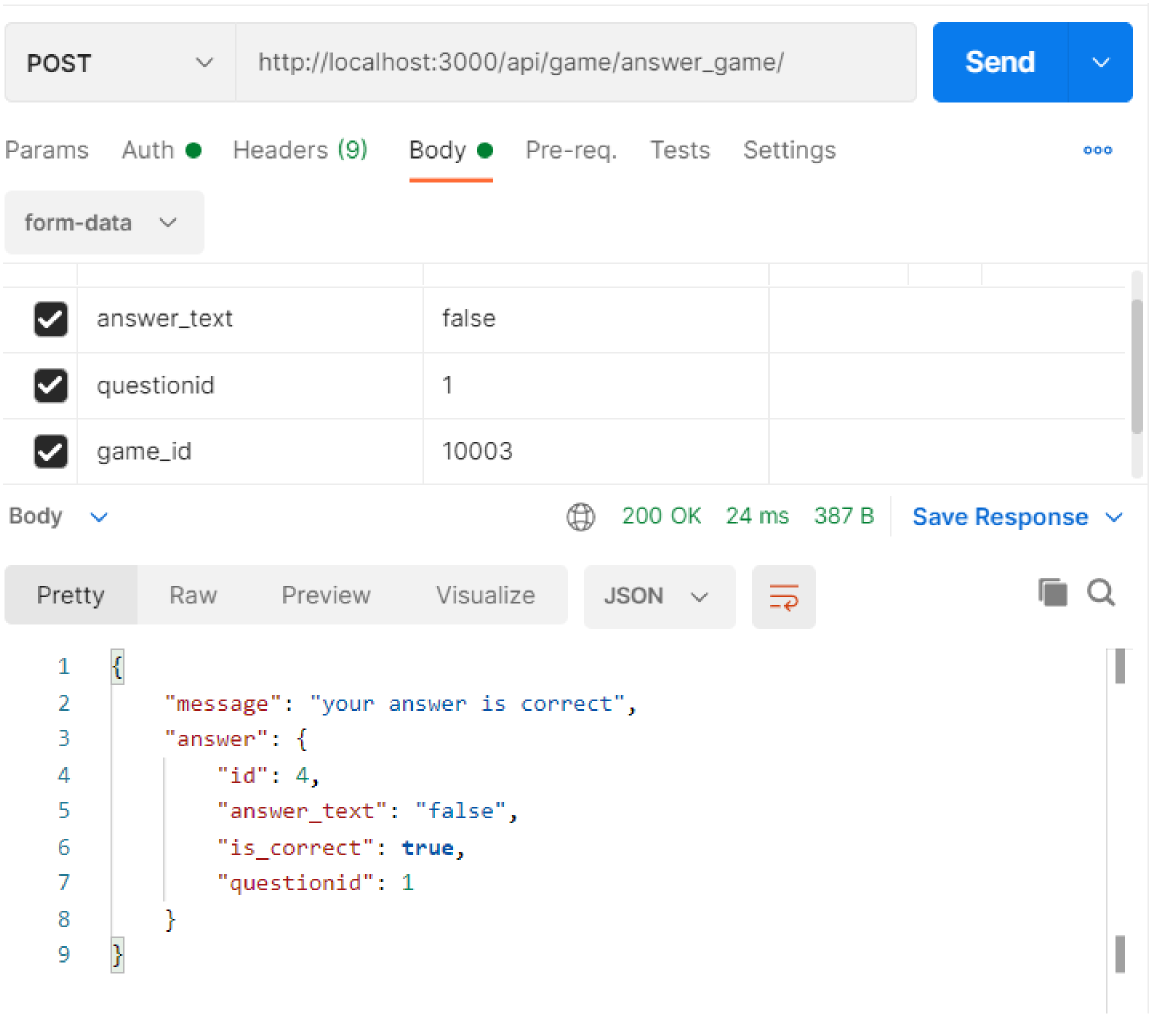
To make query on Question table by primary key is question_id, and query on Player table where game_id is the game_object that has been get from the parameter sent in the POST method and user is the current user, and query on Lobby table where primary key is game_id

```
current_user = self.request.user
player_obj = Player.objects.get(game_id=game_id, user=current_user)
obj_question = Question.objects.get(pk=question_id)
lobby_obj = Lobby.objects.get(pk=game_id)
```

Then the guesser will press right or wrong answer, and both possibility should be checked by getting guesser answer that sent in the POST method and assigning it in the answer_text object and check this answer with the correct answer that coming from Question table.Then call save() to save answer to the database.

```
if request.POST['answer_text'] == obj_question.correct_answer:
            answer = serializer.save()
                        answer.is_correct = True
                        answer.save()
                        return Response({
                        "message" : "your answer is correct",
                        "answer" : AnswerListSerializer(answer,
                        context=self.get_serializer_context()).data,
                        })
if request.POST['answer_text'] != obj_question.correct_answer:
                        answer = serializer.save()
                        answer.is_correct = False
                        answer.save()
                        return Response({
                                "message" : "your answer is wrong" ,
                        })
```

The guesser will call this class to submit his answer by endpoint path("answer_game/", api.MultiPlayerAnswer.as_view()), path in file called game/url.py in the application folder



```
http://localhost:3000/api/game/answer_game/
```

answer_text	false
questionid	1
game_id	10003

```
{
    "message": "your answer is correct",
    "answer": {
        "id": 4,
        "answer_text": "false",
        "is_correct": true,
        "questionid": 1
    }
}
```

Figure 4.2: Create or Post

Retrieve or Get

To retrieve objects from database, construct a QuerySet. To retrieve all the available games that the Player want to join it to start playing, QuerySet is needed.

To make query on Lobby table by filtering the available games in the lobby table, the filter will check:

- The games which has place for one extra new player, so the the number of players already join the game should be less that the number of players that the Lobby's creator decide.
- The games which have been made for less than one hour

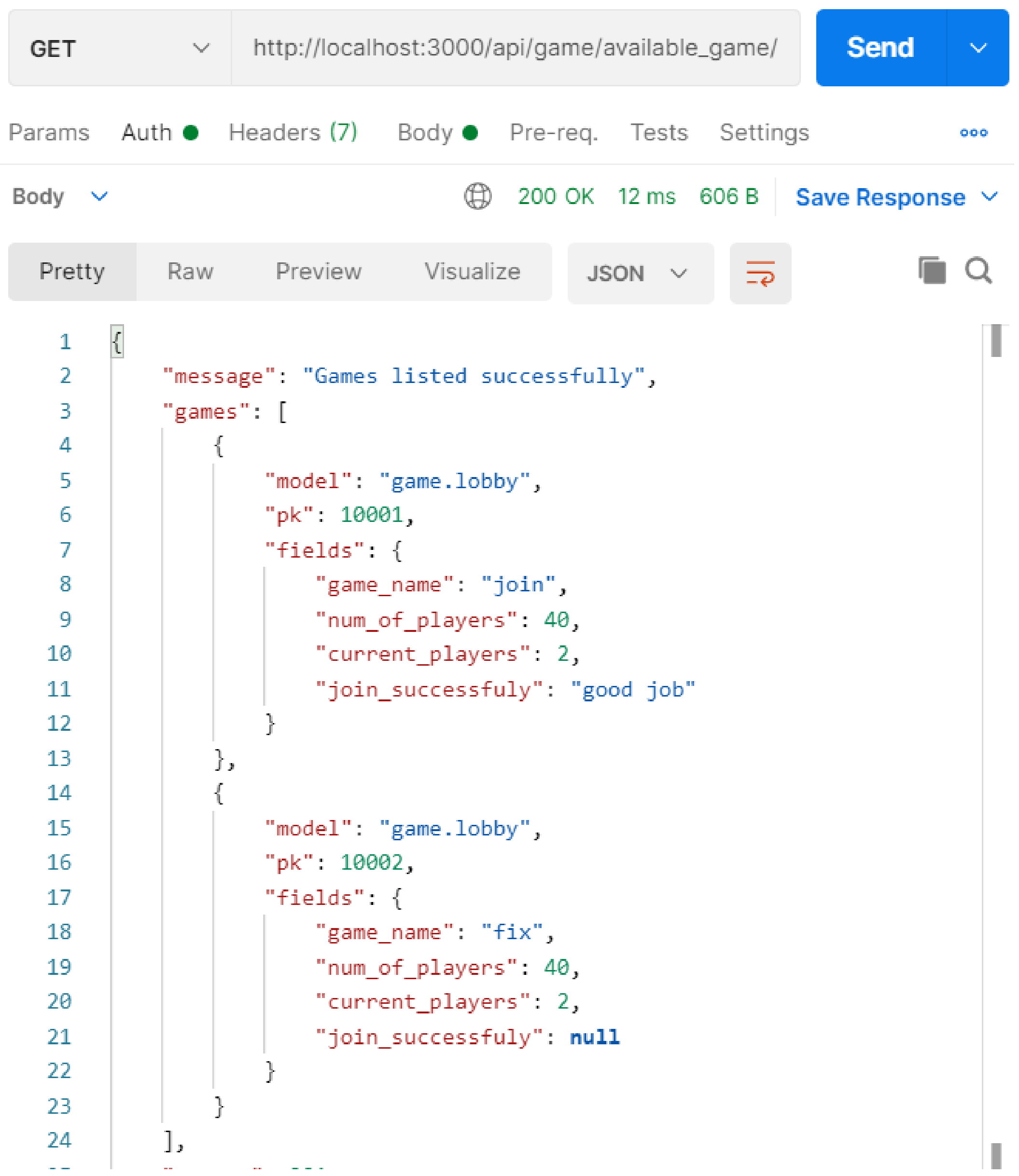
Games that meet the two conditions mentioned above, is available and the player how want to join he can find it in the frontend. F() expressions, all Python does, through Django's F() class, is create the SQL syntax to refer to the field and describe the operation.

```
class ListAvailableGames(generics.ListAPIView):
        serializer_class = LobbySerializer

        def get(self, request):
                time_diff = datetime.datetime.now() - datetime.timedelta(hours=1)
                queryset = Lobby.objects.filter(
                        current_players__lt=F('num_of_players'),
                        created_at__gt=time_diff
                        )
```

The object that queryset return it, should convert to serializable Json Object to get response with out error, and we load the converted object (tmpJson). The player will call this class by endpoint path("available_game/", api.ListAvailableGames.as_view()). path in file called game/url.py in the application folder.

```
tmpJson = serializers.serialize("json", queryset)
result = json.loads(tmpJson)
                if queryset:
                        return Response({
                  "message" : "Games listed successfully",
                  "games" : result,
                  "status": status.HTTP_201_CREATED
            })
```



```
{
    "message": "Games listed successfully",
    "games": [
        {
            "model": "game.lobby",
            "pk": 10001,
            "fields": {
                "game_name": "join",
                "num_of_players": 40,
                "current_players": 2,
                "join_successfuly": "good job"
            }
        },
        {
            "model": "game.lobby",
            "pk": 10002,
            "fields": {
                "game_name": "fix",
                "num_of_players": 40,
                "current_players": 2,
                "join_successfuly": null
            }
        }
    ],
```

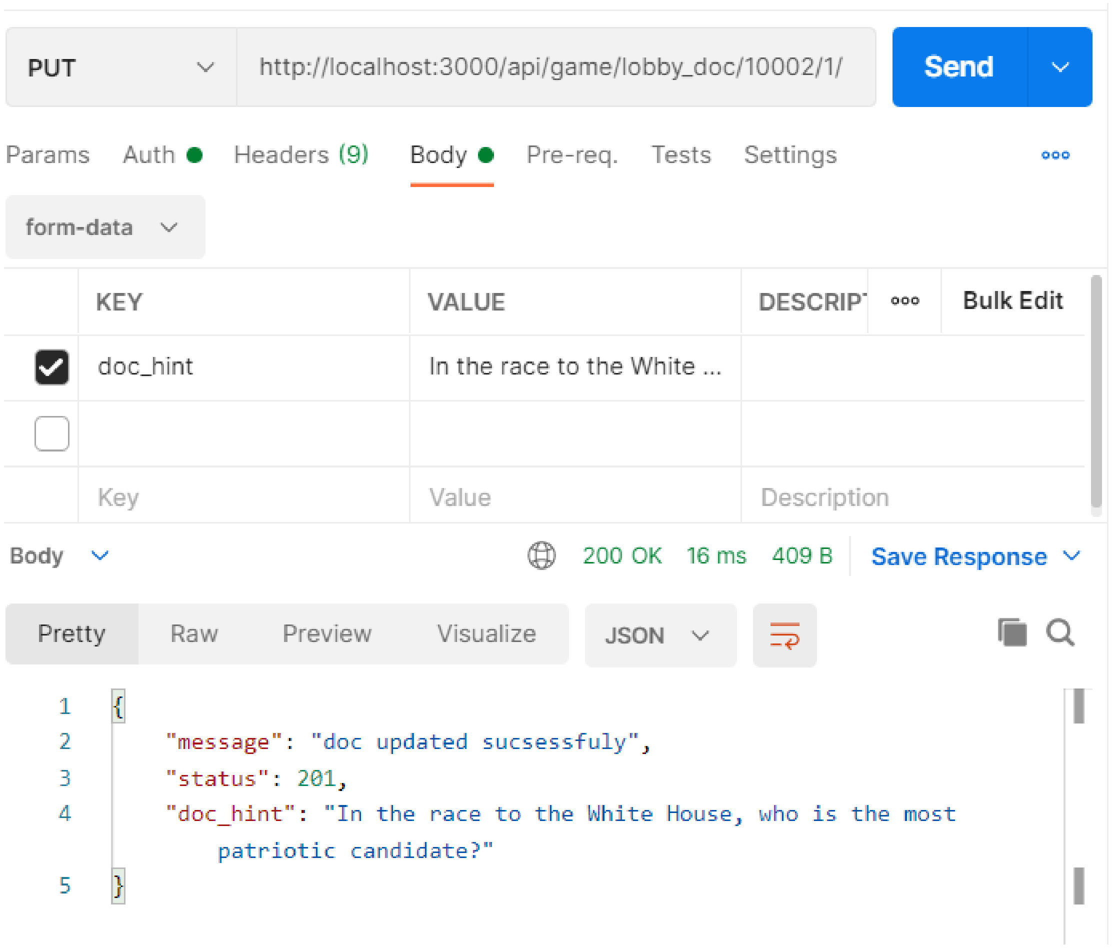
Figure 4.3: Get available games

Update or Put

To make query on Lobby table by the question_id (claim_id) and game_id. Whereas, the id of the game that the players play and the question id (claim id) that appears to the proposer will determine the field to which the sentence will be added that will help the guesser to know the correct answer. Getting the parameter sent in the POST method and assigning it into doc_hint which corresponds with question_object.

```
question_object = LobbyQuestion.objects.get(question_id=question_id,game_id=game_id)
question_object.doc_hint = request.POST['doc_hint']
question_object.save()
return Response({
        "message" : "doc updated sucsessfuly",
        "status": status.HTTP_201_CREATED,
                "doc_hint": request.POST['doc_hint'],})
```

The player will call this class by endpoint path('lobby_doc/<game_id>/<question_id>/', api.LobbyQuestionUpdate.as_view(), name='LobbyQuestionUpdate'). path in file called game/url.py in the application folder.


PUT http://localhost:3000/api/game/lobby_doc/10002/1/

Body: form-data

KEY	VALUE	DESCRIPTION
doc_hint	In the race to the White ...	

200 OK 16 ms 409 B

```
{
    "message": "doc updated sucsessfuly",
    "status": 201,
    "doc_hint": "In the race to the White House, who is the most
        patriotic candidate?"
}
```

Figure 4.4: Update or Put

4.3 Design patterns

Design patterns are used to represent the pattern used in web / API application. These patterns are selected based on the requirement analysis. The patterns describe the solution to the problem, when and where to apply the solution and the consequences of the implementation.

Structure of a design pattern The documentation of design pattern is maintained in a way that focuses more on the technology that is used and in what ways. The following diagram explains the basic structure of design pattern documentation.

Python Design Patterns Python is an open source scripting language, which is

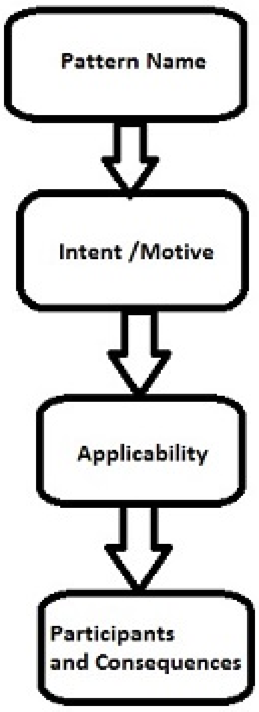


Figure 4.5: Design Pattern

high-level, interpreted, interactive and object-oriented. It is designed to be highly readable. The syntax of Python language is easy to understand and uses English keywords frequently.

Features of Python Language In this section, we will learn about the different features of Python language.

- Interpreted
  Python is processed at run-time using the interpreter. There is no need to compile program before execution. It is similar to PERL and PHP.
- Object-Oriented
  Python follows object-oriented style and design patterns. It includes class definition with various features like encapsulation, polymorphism and many more.

- Portable
  Python code written in Windows operating system and can be used in Mac operating system. The code can be reused and portable as per the requirements.

- Easy to code
  Python syntax is easy to understand and code. Any developer can understand the syntax of Python within few hours. Python can be described as "programmer-friendly"

- Extensible
  If needed, a user can write some of Python code in C language as well. It is also possible to put python code in source code in different languages like C++. This makes Python an extensible language.

Pattern Name: (Model View Controller Pattern)
Model View Controller is the most commonly used design pattern. Developers find it easy to implement this design pattern. It describes the pattern in short and effective manner.
Benefits of using design pattern:

- Patterns provide developer a selection of tried and tested solutions for the specified problems.
- All design patterns are language neutral.
- Patterns help to achieve communication and maintain well documentation.
- It includes a record of accomplishment to reduce any technical risk to the project.
- Design patterns are highly flexible to use and easy to understand.

How MVC structure works

- Model
  It consists of pure application logic, which interacts with the database. It includes all the information to represent data to the end user.

- View
  View represents the HTML files, which interact with the end user. It represents the model's data to user.

- Controller
  It acts as an intermediary between view and model. It listens to the events triggered by view and queries model for the same.

Python code:

This is an example for object called “User” and create an MVC design pattern.

Model

It calls for a method, which fetches all the records of the User table in database. The records are presented in JSON format.

Algorithm 4.1 model.py

```
import json
class User(object): def ___init___(self, first_name = None, last_name = None):
self.first_name = first_name
self.last_name = last_name
#returns User name, ex: John Doe
def name(self):
return (" (self.first_name,self.last_name))
```

$\frown classmethod$

$\#returnsallpeopleinsidedb.txtaslistofUserobjects$

$defget\_all(self):$

$database = open('db.txt','r')$

$result = []$

$json\_list = json.loads(database.read())$

$foriteminjson\_list:$

$item = json.loads(item)$

$user = User(item['first\_name'], item['last\_name'])$

$result.append(user)$

$returnresult$

View

It displays all the records fetched within the model. View never interacts with model; controller does this work (communicating with model and view).

Algorithm 4.2 view.py

```
from model import User def show_all_view(list):
print 'In our db we have
for item in list:
print item.name()
def start_view():
print 'MVC  the simplest example'
print 'Do you want to see everyone in my db?[y/n]'
def end_view():
print 'Goodbye!'
```

Controller

Controller interacts with model through the **get_all()** method which fetches all the records displayed to the end user.

Algorithm 4.3 api.py

```
from model import User import view
def show_all():
#gets list of all User objects
people_in_db = User.get_all()
# calling view
return view.show_all_view(people_indb)

def start():
view.start_view()
input = raw_input()
if input == 'y':
return show_all()
else:
return view.end_view()

if ___name___ == "___main___":
# running controller function
start()
```

Chapter 5

Frontend Mobile Application

5.1 Introduction

Mobile application or mobile app for short is a computer software or software program designed to run on mobile devices such as a phone, tablet or smartwatch. Mobile applications are usually downloaded from application distribution platforms like App Store or Google Play Store. Mobile apps were originally created to offer general productivity and information retrieval like email, calendar, contacts, the stock market and weather information. However, the public demand and the rise of popularity of mobile applications led to a rapid expansion into other categories. Today, a smartphone have practically all the functions of a computer thanks to all the different mobile applications that have been created and studies have showed that more people tend to use apps rather than web browsers when browsing the net. Researchers found that the use of mobile apps strongly correlates with user context and their whereabouts and time of the day.
Mobile applications are often categorised in three categories: Native apps, Hybrid apps and Web-based apps.
Native apps are apps that are targeted toward a particular mobile platform and therefore only exclusive to that mobile platform. The main purpose of creating such apps is to guarantee the best user experience can consistency for that specific mobile operating system.
Hybrid apps are a mix of native and web-based apps. These apps are made to support both web and native technologies across multiple platforms. They also tend to be both faster and easier to develop than their counterparts because of their use of single code base which work in multiple mobile operating systems.
Web-based apps are coded in HTML, CSS or Javascript and require internet to function. Web-based apps require less memory space compared to their counterparts since all the

personal databases are saved in the Internet servers.

5.2 Choice of mobile OS and framework

When creating a mobile application, it is desirable to create a mobile app that works for as many mobile operating systems as possible. Android is the most used mobile OS, having a market share of 71.81% worldwide. The second largest market share, iOS has a market share of 27.43%. Using a framework like Flutter, that supports both iOS and android apps, the mobile application reaches up to 99.24% of mobile OS usage, thus making the app available for most mobile users worldwide.

5.3 Script functionalities

The frontend is written by flutters own language, Dart and is responsible for creating and displaying the data from the server to the UI when the mobile application is in use. The frontend consists of a login, register, home page and settings plus game modes like singleplayer and multiplayer.

On startup, the application start at the login site where the user is able to register themselves and log in to the game where they enter the home page. The homepage consists of a game mode page, settings page. In the game mode page, the user can choose between playing single player mode or multiplayer mode.

5.3.1 Login and register

The login site is the first site the user sees when the user start up the app. The login site consists of a username input and a password input the user can fill in if he have an existing account. If the user does not have an account, he can press the register button where he is then redirected to the register page which consists of a form the user have to fill in. The form has a username, email, password and confirm password field. Each field have a certain requirement that has to be filled for it to pass. If one of the input requirements are not filled, an alert box will be displayed for the user informing them of what input is wrong and has to be changed in the register form. All fields have to be filled for the account to be created.

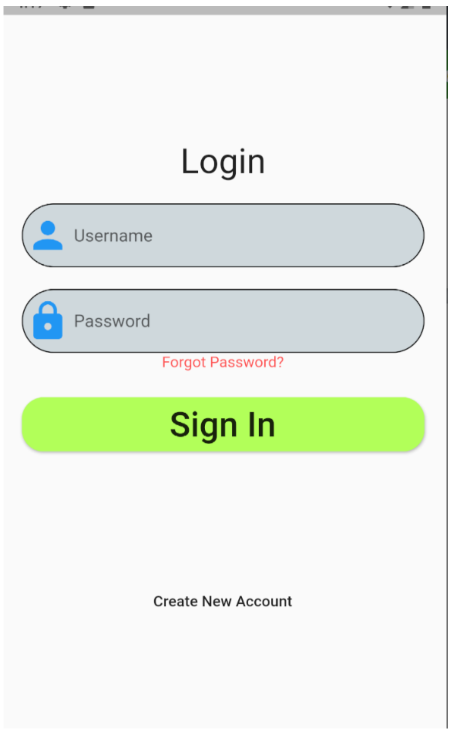


Figure 5.1: Login page

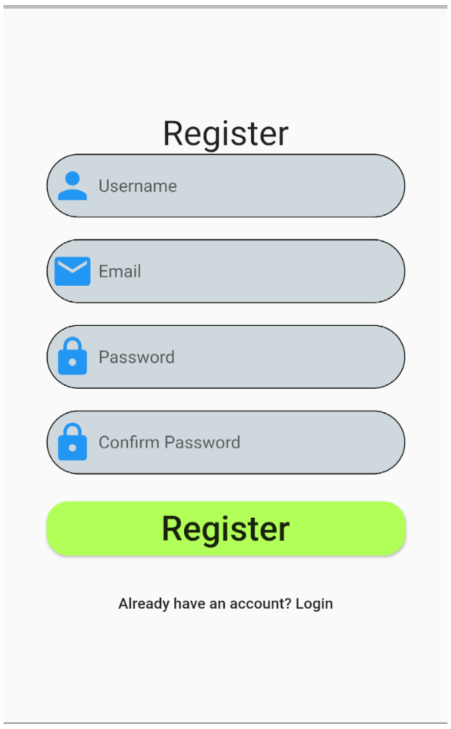


Figure 5.2: Register page

Algorithm 5.1 Login

```
0: procedure LOGIN($username, password$)
0:   databaseHelper ← DatabaseHelper()
0:   _usernameController ← username
0:   _passController ← password
0:   if $\_usernameController.isNotEmpty \&\& \_passController.isNotEmpty$ then
0:     databaseHelper.login ← _usernameController, _passController
0:     if $databaseHelper.errorMessage$ then
1: return $error$
1:     else if $databaseHelper.errorMessage \neq true$ then
1:       Redirect to Homepage
1:     else
2: return $error$
   =0
```

5.3.2 Home page

Home page is the main page that connects all the other pages together in the mobile application. The homepage consists of a game mode tab and an Options tab. In the game mode tab, there are three buttons: singleplayer, multiplayer and how to play. Each single button redirects their respective pages. The options tab have a logout button that upon pressing will redirect the user back to the login page. Login page, back button on

Algorithm 5.2 Register

```
0: procedure REGISTER($username, email, password, confPassword$)
0:   databaseHelper ← DatabaseHelper()
0:   _usernameCtrl ← username
0:   _emailCtrl ← email
0:   _passCtrl ← password
0:   _confPassCtrl ← confPassword
0:   if $\_usernameCtrl \&\& \_passCtrl \&\& \_emailCtrl \&\& \_confPassCtrl$ then
0:     databaseHelper.registerData ← _usernameCtrl, _emailCtrl, _passCtrl
0:     if $databaseHelper.errorMessage \neq null$ then
0:       $Createaccount$
0:     else if $databaseHelper.errorMessagenull$ then
1: return $Msg$
1:     else
2: return $error$
   =0
```

the start game page, leave lobby button on the waiting lobby page and endscreen page all redirects the user to the Home page.

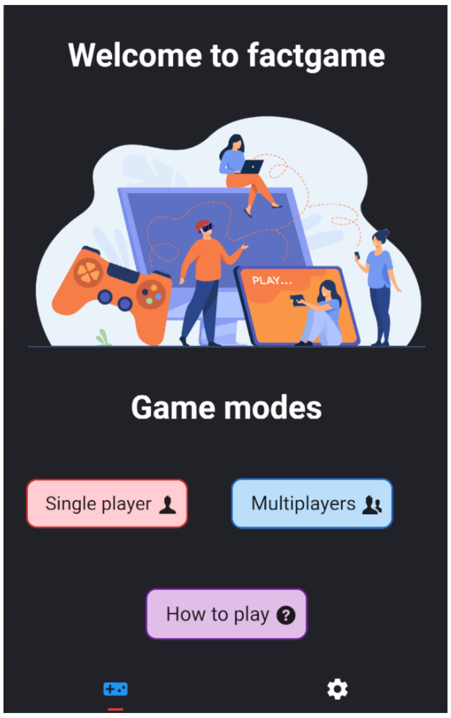


Figure 5.3: Game mode page

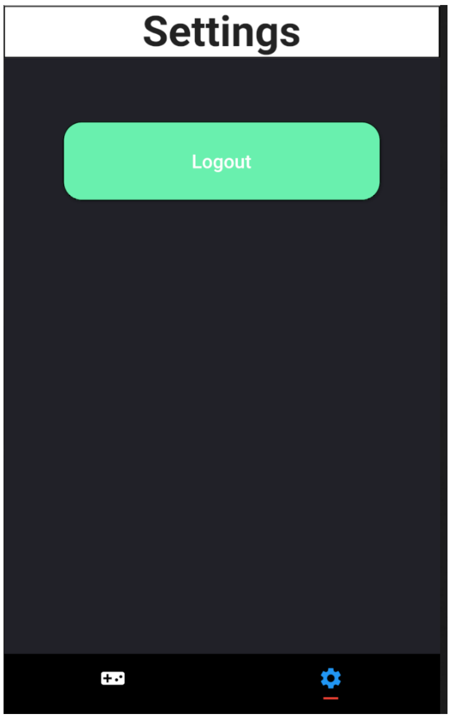


Figure 5.4: Settings page

5.3.3 Singleplayer

The singleplayer consists of seven functions as well as a widget that makes up the UI. In the singleplayer game mode, the player plays with the program. The claims displayed to the player is picked at random from the database by the program.The player is given 10 claims that the player have to guess whether is true or false based on the evidence given by the program. The set of answers the user can pick between are:

- True
- Barely true
- False
- Mostly true
- Pants on fire
- Half true

If the player does not manage to find the answer fit for the claim, the score will not change, but if the player manage to answer correctly, the score will be increased by:

$$score = 10 * timer \qquad \text{"timer" is the amount of time the user has left}$$

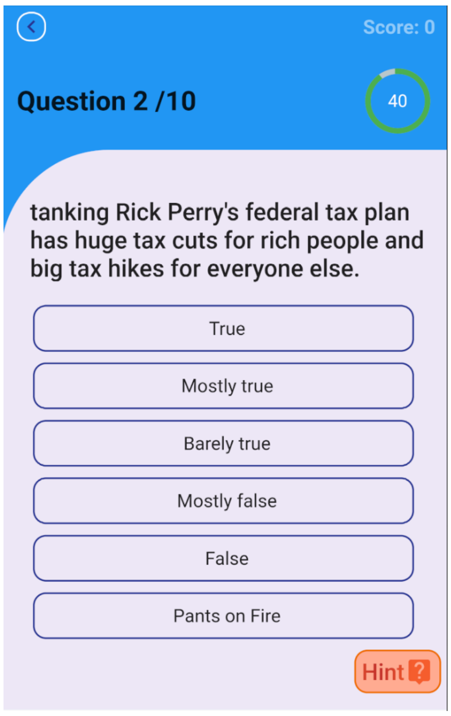
The functions in the singleplayer game is:

- FitchData
- Shuffle
- ShuffleHint
- ShowQuestion
- nextQuestion
- checkanswer
- starttimer

The function FitchData is a function that retrieves metadata from the database that is later used to creates variables for the singleplayer game to use. FitchData is ran upon entering the singleplayer page and will run shuffle and showQuestion after collecting the data.
Shuffle is a function that shuffles the order of the datalist that is retrieved from FitchData. This is done so that the questions that is displayed in the UI is randomized. ShuffleHint does the same thing except that it just shuffles the list hints for each claim it is attached with.
ShowQuestion runs upon start of the page as well as whenever the user has pressed the button NextQuestion. ShowQuestion uses an if loop that runs up to 10 times as that is the amount of rounds we want the user to play. In the if loop, claims, sources, hints and answers are retrieved from the map created in FitchData and creates the variables:

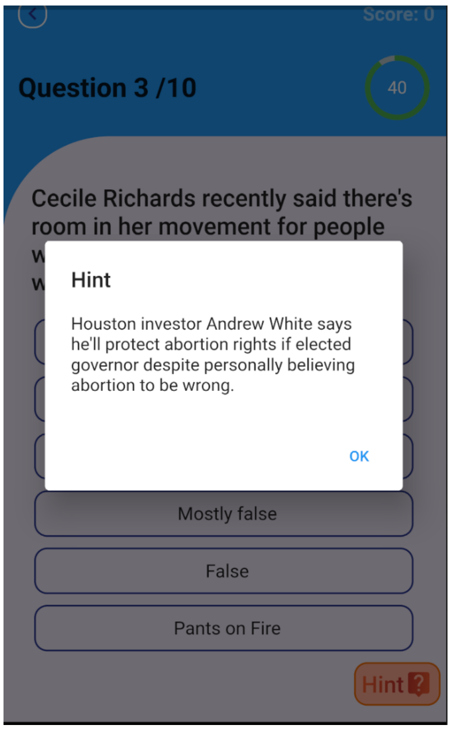
- Claim
- Answer
- Hint
- Source

Each of these variables are then either displayed in the UI or used in another function.
The nextQuestion function is the function that updates the question variable for the application to display in the UI as well as resets all changes made in the UI such as the changing the colour button back to its default button and resets the timer back to 45 seconds. This function runs on startup of the page as well as whenever the user clicks the next question button.
The checkanswer function is called upon whenever the user have pressed one of the available answers displayed in the UI. Upon clicking the answer, a string is then sent to the checkanswer function where it is compared with the correct answer of the claim. If the answer is correct, the score will update and the button the user clicked will turn green, but if the user answers wrong, the button will instead turn red.



tanking Rick Perry's federal tax plan has huge tax cuts for rich people and big tax hikes for everyone else.

Figure 5.5: Singleplayer page UI



Cecile Richards recently said there's room in her movement for people w w

Houston investor Andrew White says he'll protect abortion rights if elected governor despite personally believing abortion to be wrong.

Figure 5.6: Hint displayed for guesser

Algorithm 5.3 Singleplayer Guesser

```
procedure SINGLEPLAYER GUESSER
  mapResponse ← json.decode(response.body)
  question ← first question in map of shuffled mapResponse
  questionid ← first questionID in map of shuffled mapResponse
  answer ← first answer in map of shuffled mapResponse
  score ← 0
  timer ← 45
  canceltimer ← false
  colortoshow ← Colors.indigoAccent;
  right ← Colors.green;
  wrong ← Colors.red;
  if $answer = question$ then
    procedure $checkanswer()$
      $canceltimer$ ← true
      $colortoshow$ ← right
      $score$ ← 10*timer
      procedure $nextquestion()$
        $colortoshow$ ← Colors.indigoAccent;
        $timer$ ← 45
        $canceltimer$ ← false

        procedure $checkanswer()$
          $canceltimer$ ← true
          $colortoshow$ ← wrong
          procedure $nextquestion()$
            $colortoshow$ ← Colors.indigoAccent;
            $timer$ ← 45
            $canceltimer$ ← false

```

5.3.4 Multiplayer

In multiplayer, the user can either choose between creating their own lobby or join an existing lobby. If the user choose to create their own lobby they have to first choose how many players is going to play and the name of the lobby. At default, the lobby has a size of two, one guesser and one proposer. After the lobby has been created, the user who created the lobby gets to choose what Statements the guesser has to debunk.

The multiplayer game mode consists of five pages; createlobby, joinlobby, waitinglobby, guesser and proposer. Each page have their own functionalities that makes up the multiplayer game mode for the application.

The createlobby page conists of two input lines, one for lobby size and one for lobby name. In this page, the user is able to create a lobby simply by inserting lobby size and lobby name. After filling in the input lines and pressing "create lobby", the data that the user have given the application from the UI is then proccessed in the backend and creates a lobby for other players to see. After creating the lobby, the user is then automatically redirected to the waitinglobby page, of which is a page specifically created that lobby. In the waitinglobby page there are two buttons in the UI that the user can click, Start game and leave lobby. Clicking either one of them will either redirect them to the proposer page or guesser page depending on what role the user has, or be redirected back to the homepage.

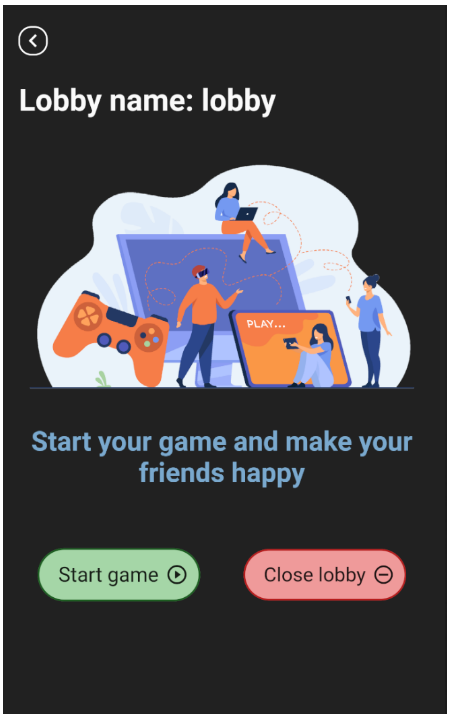


Figure 5.7: Waitinglobby

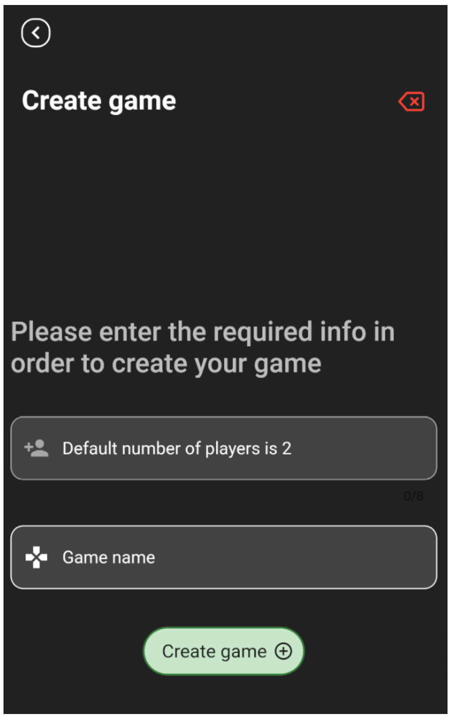

Create game
Please enter the required info in order to create your game
Default number of players is 2
0/8
Game name
Create game

Figure 5.8: Createlobby page

The joinlobby page is a page that consists of a list of lobbies that has been created by other players. Clicking on one of the lobbies, will send data to the backend and inform the database that the user wants to join said lobby. The user is then redirected to the waitinglobby page of that lobby. Users gets their role in the multiplayer game mode when joining the lobby. If the user created the lobby, they will receive the role of proposer and will be redirected to the proposer page when they press "start game" in waitinglobby. If the user joins a lobby through the joinlobby however, they will receive the role of guesser and will be redirected to the guesserpage upon starting the game. As guesser, the user have to guess the fitting answer to the statement given. The score system is like the one of singleplayer where if you answer correct the score is increased by:

$$10 * timer$$

The user will not lose any points if the answer is wrong.

As proposer, the user has a selection of hints, the user can choose from to display to the guessers. The proposers job is to get as many guessers to answer correct. The proposers score is calculated by:

$$\frac{10 * collectiveGuesserScore}{totalPlayers}$$

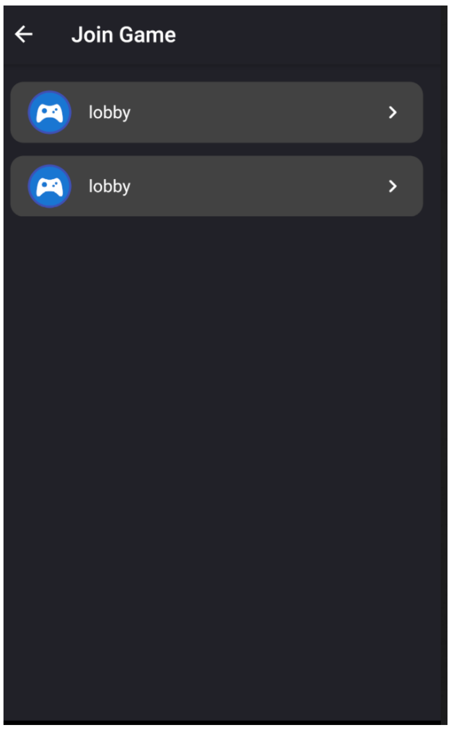
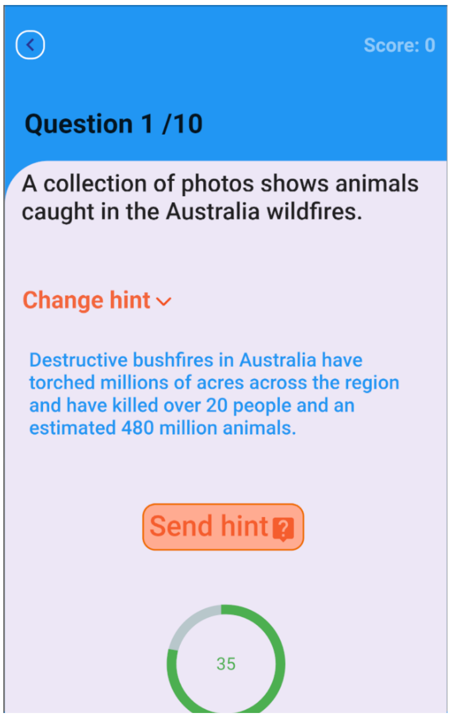


Figure 5.9: Joinlobby page



A collection of photos shows animals caught in the Australia wildfires.

Destructive bushfires in Australia have torched millions of acres across the region and have killed over 20 people and an estimated 480 million animals.

Figure 5.10: Proposer page

5.4 Communication between scripts

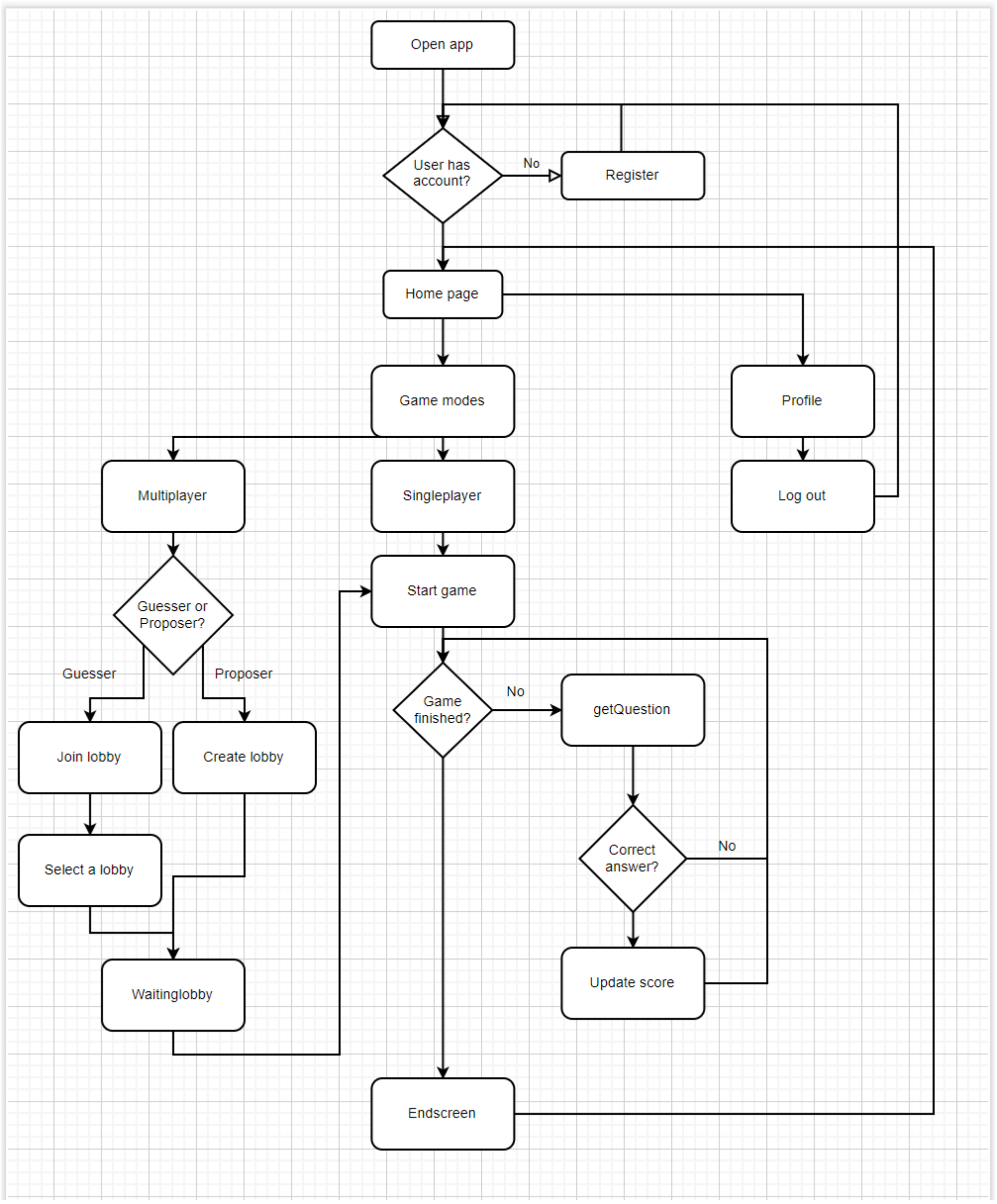


Figure 5.11: Flowchart

Chapter 6

Experimental evaluation

6.1 Introduction

This chapter shows the results from four users playing both the singleplayer game mode and the multiplayer game mode. Average game time, average score, how many correct answers they got as well as user feedback is documented.

6.2 Singleplayer results

In the singleplayer game mode, the user is tasked to detect fake news out of the given 10 claims with the help from the program through hints the program have chosen. In the singleplayer game mode, the average completion time was 3 minutes and 25 seconds, meaning an average of 20,5 seconds per question. The average score in the singleplayer game mode was 630. The highest score documented was a score of 2200 with a time of 2 minutes and 36 seconds, which also was the fasted time recorded. It was recorded two games where the user was unable to answer correctly in any of the claims and therefore ended up with a score of 0. The charts below shows how the score is distributed between each playthrough and the amount of correct answers.

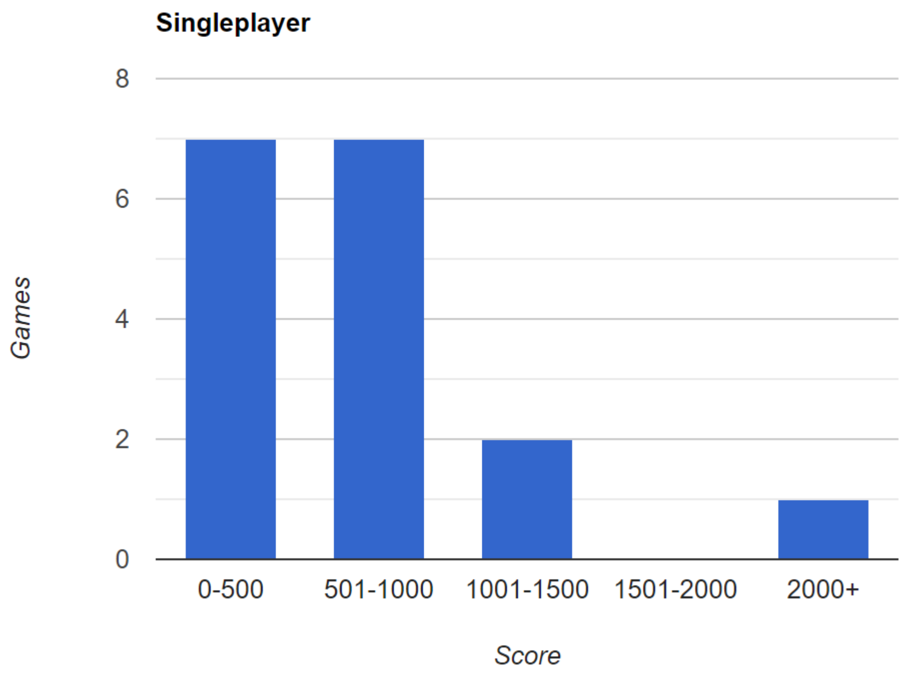


Figure 6.1: Bar chart of singleplayer score

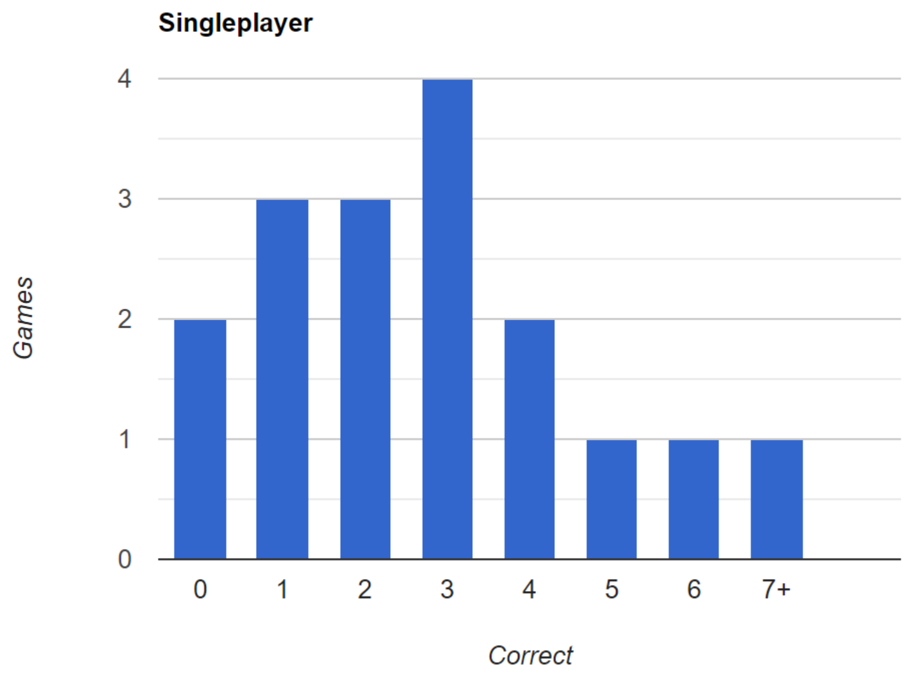


Figure 6.2: Bar chart of singleplayer correct answers

6.3 Multiplayer results

In multiplayer game mode, we have two roles, guesser and proposer. The guesser is tasked to try and guess whether given claims are true or not, while the proposer is tasked to help the guesser out by giving the guesser hints the proposer believes will help out the guesser. In the multiplayer game mode, the average game time was 3 minutes and 12 seconds, with an average score of 820. This means that the average time used and score per claim was 19,2 seconds and 82 respectively. In multiplayer, it was only recorded a zero point game once, while the highest score was recorded at 2400 with a time of 3 minutes and 52 seconds. The charts below shows how the scores were distributed between each game as well as the amount of correct answers the guesser got.
Note that in this experiment, the proposer and guesser got the same score as only two players were playing together in each game meaning that they got the same score and therefore the scores below shows the scores gathered from each lobby instead of each player.

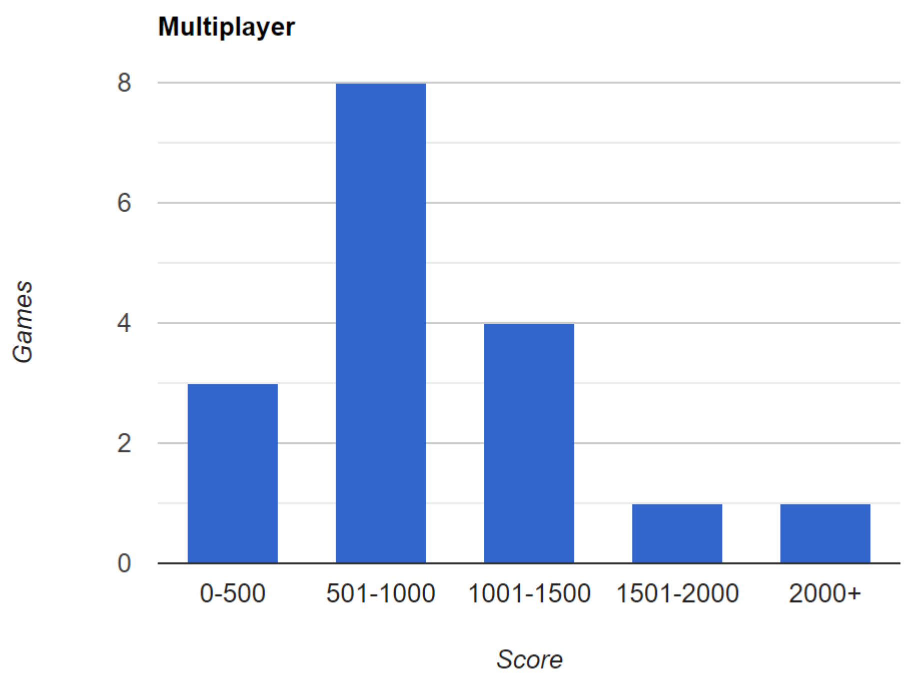


Figure 6.3: Bar chart of multiplayer score

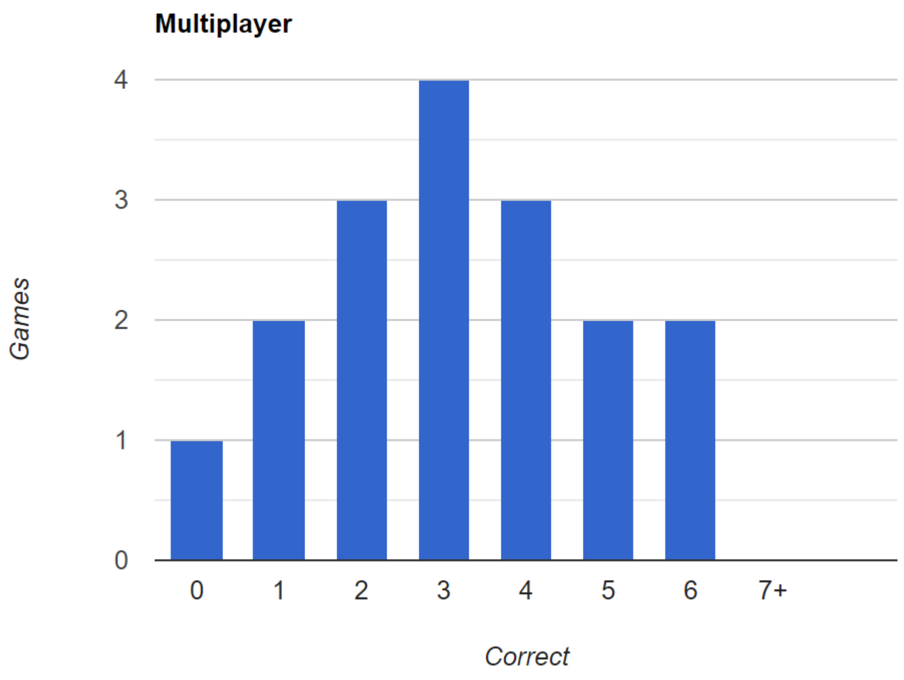


Figure 6.4: Bar chart of multiplayer correct answers

6.4 User feedback

Some of the most common problem the players had when playing the game was the fact that most if not all of the claims where USA related, something not many of the players where informed about. Another problem was the fact that in singleplayer, the claims where sometimes too bad for the players to be able to determine whether the claim was true or not. It was also really hard to differentiate a claim from being for example "false" or "pants on fire" and "true" , "half true" or "mostly true" where often punished because of it. In multiplayer, most of those problem where the same but not all of them. For instance, in multiplayer, the proposer was able to pick the hint he thought was the best available hint, while in singleplayer it was still a variety between good and bad hints. Such results are easily seen in the bar charts above (figure 5.1 and 5.2), where the average score in multiplayer is higher.
A thing that one of the players noticed was that whenever the hint contained facebook or had its source from facebook, the answers where always "Pants on fire". Hints that did not contain facts or statistics where usually never "True" according to some of the players. The same went for whenever the hint contained a quote from the person in question.

Chapter 7

Discussion

7.1 Problems and Challenges

7.1.1 Handle Huge data

A problem is that from the hundreds of claims we were able to gather from https://www.politifact.com/, we were only able to fit 81 of those claims with their answer, sources, hints, etc. This caused the game to be quite repetitive as the same claims would pop up after a couple playthroughs. This also caused the answers to the different claims to be badly distributed to the point where most of the answers were false.
While we were able to increase the to fit more claims into the database by buying more space, we stumbled upon a new problem. This time it was an optimization problem with the OS. Because the program have to retrieves all the data attached with claims from the database to shuffle it before displaying it in the UI, and the fact that the database contains over 1000 claims, shuffling the claims and displaying it ends up taking a considerable amount of time. The time it takes for the program to process the claims, shuffling them and picking out 10 out of the 1000 claims heavily depends on the devices CPU.

7.1.2 Testing and Production ready app

Another problem was to publish the application for testing purposes. Application could not go live unless there is a "domain" , "host / server" and "An SSL certificate" installed on the host. That was the only option to solve the "origin cors" bug for building deploying the game on "Android iOS" devices.
This however was more of a budget problem as publishing the application for Android

would cost $25 and $99 yearly for app store. This made it very hard to get data for the experimental evaluation as users would have to pull our code from "github" repository as well as download "Android studio IDE" and "Flutter" for the tester who use android devices, but for the tester how use iOS need to download *xcode IDE* in addition to "Android studio IDE" and "Flutter".
This made installing the game quite unappealing as it is too much work.
We also tried to publish the application as a web application in heroku just for the testing purposes, but that would require a lot of editing in the code as well as downloading a lot of other extensions. While we did manage to publish it to heroku, we stumbled upon a problem where the application did not retrieve any data from our web server and therefore was just an empty application.

7.1.3 MacBook X-code

To test the application on iOS system we need MacBook, both of us has windows system which is work with android operating system. We try to borrow on MacBook from a friend and after we download "Android studio IDE" and "Flutter". we start to download *xcode IDE* to run the *Apple* simulator, but we found that there is no enough space in the MacBook that we borrowed because *xcode IDE* take space more than 30 GB disk space, so we start searching for a friend can help us to test our App until we found.

7.1.4 Learning flutter/dart

A challenge was learning and using flutter for the first time. As none of the group members had any experience with flutter, we had to use the first couple of weeks learning the basics of dart. Despite using our time in learning the fundamentals of flutters we still stumbled upon new specific problems with flutter throughout the project. However, learning this new language and how to create our first ever mobile app was something quite interesting and a fun challenge for us both.

7.1.5 build database and github

One of the problems was transferring data from localhost to the server. Every time we modify the database, must delete many Django files related to the database and delete all the tables from the data base, and then must build the database again and push this new data to repository `https://github.com/karamyanes/fakenews-app.com`. To connect to server Putty was best solution and there we have to pull the data that repository has

it. In Putty we should pull the new data and every change in database's tables, the old Docker's container should be delete and new Docker's container need to be build. All that job consume lot of time even if the part changed in database is on word.
Here between pull and push it was many git message's errors coming, which led us to study and read more about git.
On the other side when me and my partner make changes on the same file and push it to repository that led to merge and lot of job to fix this merge and keep all changes that have been added.

7.1.6 Handle new request: Word Tokenizer

Another challenge was an additional task given by our supervisor. In late April, the supervisor suggested that we should try to optimize the hints given from the program in single player, by using a tokenizer. As we previously had shuffled all the hints for the claims, the hints sometimes were irrelevant or had no relation to the claim itself. With completing the new task, the tokenizer should sort the hints related to each claim and divides a text into a list of sentences given to the players so that whenever the player wants a hint, the hint given should be a relevant hint rather than a complete random one.

7.2 Further work

7.2.1 Automation and Deployment

One of the good to have features in any development process is to have an automated deployment process from code on local machines to development branch on "Github" to "Testing" branch to "Master" branch.
Then auto deploy the code with the new changes from master to server.
Adding the application to app store or google play is the one thing that is missing for it to be easy accessible for the mass. However, it is still accessible for anyone to try out by first download flutter and android studios (+ xcode if you have iphone) and then pull the application from GitHub and run it on android studios while having your phone connected to the PC.

7.3 Conclusion

In this project, a mobile application game about fake news was made. The application is able to run in both apple and android devices which roughly makes up 99.24% of users worldwide, thus making it accessible for practically everyone.
The goals of the project was to create fun way of learning about fake news in a from of a game with a proposer and a guesser, as well as retrieve data from the users and track their performances and what they thought was important in detecting fake news. This multiplayer game is built in such a way that both sides were presented with a claim and where the proposer had to help the guesser by giving the guesser hints from a list. A web server was set up using `https://www.digitalocean.com/` so that both the proposer and the guesser is able to retrieve the same data at the same time. The data that was then inserted into the webserver for the mobile application to display was retrieved from `https://www.politifact.com/` using an API retriever created/found by the supervisor. Creating the mobile application itself was done using flutter and its built-in widgets. A singleplayer game mode was also created where the player played as a guesser with the program who played as the proposer.
The application is able to be run in both an emulator and in a real life device by downloading the file from the PC into the device itself. All that is needed for the application to be run smoothly is having internet connections as all the data used in the application is in a web server. There are still some minor optimizations that could be done before the application could be released in app store or google play.

List of Figures

Bibliography

[1] wikipedia. Fake news, Retireved March 2021. URL https://en.wikipedia.org/wiki/Fake_news.

[2] dictionary.com. Fake news, Retireved March 2021. URL https://www.dictionary.com/browse/fake-news.

[3] Joyce Rice Chas Brown Kelli Dunlap Cherisse Datu Lindsay Grace Amy Eisman Maggie Farley, Bob Hone. Factitious, Retireved April 2021. URL http://factitious.augamestudio.com/#/.

[4] DROG. getbadnews, Retireved April 2021. URL https://www.getbadnews.com/#intro.

[5] Amanda Warner. fakeittomakeitgame, Retireved April 2021. URL https://www.fakeittomakeitgame.com/.

[6] Inventionland. The history of mobile app. URL https://inventionland.com/inventing/the-history-of-mobile-apps/.

[7] BBC. A brief history of fake news. URL https://www.bbc.co.uk/bitesize/articles/zwcgn9q.

[8] April fool's day. URL https://www.dw.com/en/belgium-police-clash-with-april-fools-day-party-crowd/a-57083528.

[9] Wikipedia. Game theory, Retireved April 2021. URL https://en.wikipedia.org/wiki/Game_theory.

[10] American Experience. Game theory explained. Retireved April 2021. URL https://www.pbs.org/wgbh/americanexperience/features/nash-game/.

[11] Adam Hayes. Game theory. Retireved April 2021. URL https://www.investopedia.com/terms/g/gametheory.asp.

[12] Swagger open source. URL https://swagger.io/tools/swagger-ui/.

[13] Swagger. Swagger open source. URL https://searchapparchitecture.techtarget.com/definition/Swagger.

[14] Postman. The collaboration platform for api development, Retireved 2021. URL https://en.wikipedia.org/wiki/Game_theory.

[15] Koder. Django rest api. URL https://www.computerhope.com/jargon/p/putty.htm.

[16] Beautiful soup documentation. URL https://www.crummy.com/software/BeautifulSoup/bs4/doc/.

[17] Target internet. What is data scraping and how can you use it? URL https://www.targetinternet.com/what-is-data-scraping-and-how-can-you-use-it/.

[18] Mike Stowe. Learn api, December 04, 2014. URL https://blogs.mulesoft.com/dev-guides/api-design/api-best-practices-nouns-crud-etc/.

[19] Django. Djangothe web framework for perfectionists. URL https://docs.djangoproject.com/en/3.2/topics/db/models/.

[20] Django Documentation. Making queries. URL https://docs.djangoproject.com/en/3.2/topics/db/queries/.