School of GeoSciences

Dissertation
for the degree of

MSc in Geographical Information Science

Lalehsadat Moussavi

August 2018

Story Telling in Space and Time:
A Mobile Application

Statement of Copyright

Statement of Originality

I declare that this dissertation represents my own work, and that where the work of others has been used it has been duly accredited. I further declare that the length of the components of this dissertation is 5409 words for the Research Paper and 6425 words for the Technical Report.

8th August 2018

Acknowledgements

From the staff of the University of Edinburgh, I would like to firstly thank my supervisor Dr. William Mackaness for his guidance throughout the project. I would also like to thank Owen Macdonald for his assistance in writing of the dissertation.

My deepest thanks to my husband and my parents for their unwavering support.

Part I

Research Paper

Table of Contents

1 Abstract

Walking around a city in a group with a human tour guide has long been the primary way of learning about a city. However, advances in the Global Positioning System have enabled walking tour mobile apps to be increasingly seen as a viable alternative (Festa, 2016, Sui and Goodchild, 2011).

Within the realm of Location-Based Services (LBS), several examples of walking tour mobile apps already exist (Lu and Arikawa, 2013), but their applicability is still limited as they are typically focused on a specific technique to help the user find his/her way. This dissertation project instead aims at using multiple techniques to produce a ubiquitous storytelling walking tour app, named Tour Guide. Tour Guide has an intuitive design to provide support to the user in finding his/her way and help the user cope with getting lost, similar to how a traditional human tour guide mentors a group. Using storytelling techniques make listening to the content fun, intuitive design makes the app user-friendly and the app's ubiquity features make it easier for the user to put her phone away and immerse herself in the surroundings instead of staring at the screen.

A proof-of-concept mobile Android app is developed for a ghost tour in the Royal Mile street of Edinburgh, Scotland. Tour Guide is undergone user testing in the field. Results are outstanding and show that the app is successful in attracting the users' positive opinions. Despite the fact that none of the users had any mobile walking tour experience before, most of them now prefer using mobile app tour rather than a human tour.

Keywords: Walking Tour App, Geofencing, Storytelling, Android Application, Location Based Services, The Royal Mile Edinburgh, Ghost Tour

The Royal Mile's Ghost Tour Walking App is available at: https://github.com/LalehMoussavi/storyTelling

2 Introduction

Imagine that you are walking around a city and are curious about what happened on this very street you are walking on or to the people who lived here before. One way is to go on a tour with a human tour guide who takes you to places and brings them alive with stories of the dead. Human tours are traditionally very appealing; however; they are normally organized in groups. The group assembles at a specific time. It moves slowly, because everybody has to gather up around the guide when s(he) is narrating a story. If one needs to stop, e.g., to have a quick break or take a photo, s(he) will fall behind. This is often quite an unpleasant experience that ends all the beginning excitements (VoiceMapLtd, 2018a).

The prevalence of smart phones equipped with built-in GPS and internet access enables us to rely on an alternative approach: a mobile application tour guide (Clark, 2011). The story has a narrative, the journey has a route, the two are zipped together, and the stories take shape (Mackaness, 2012).We can use smart phones to tell situated stories so that we learn about the city as we travel through it just similar to a traditional human tour, but not with its attached restrictions.

Using an app for walking tour is considerably less expensive than the traditional option (Kang et al., 2017) and offers a great sense of autonomy to the user since the itinerary can be tailored to the his/her need (Dickinson et al., 2014). A walking tour app minimizes the effort required to fit in the human tours' inflexible schedules. Thereby leaving users free to enjoy the tour at their own desired time and pace rather than sticking to the human tour guide (Owaied et al., 2011).

The goal of this thesis is to develop a mobile application tour guide. To focus efforts on the project, the following research questions were asked:

- Can we use smartphone technology to guide us through the streets and tells us situated stories that take into account where we have just been, and where we are? In other word, can we have streetstorytell instead of streetview? (Mackaness, 2012)
- Is the android the right programming environment for creating the app?
- How well the app can perform the tour, versus a traditional human tour?

While the ultimate ambition of a project like this would be to make a walking tour app for anywhere which has a story to tell, the scale of this proof of concept project was limited to a case study area – Ghost Tour of the Royal Mile street in Edinburgh, Scotland (Figure 1). The app, however, is simply extendable to other tours and other places.

Figure 1: The Royal Mile street of Edinburgh, Scotland, shown with red line. It starts from the Edinburgh castle and ends at the Holyrood palace

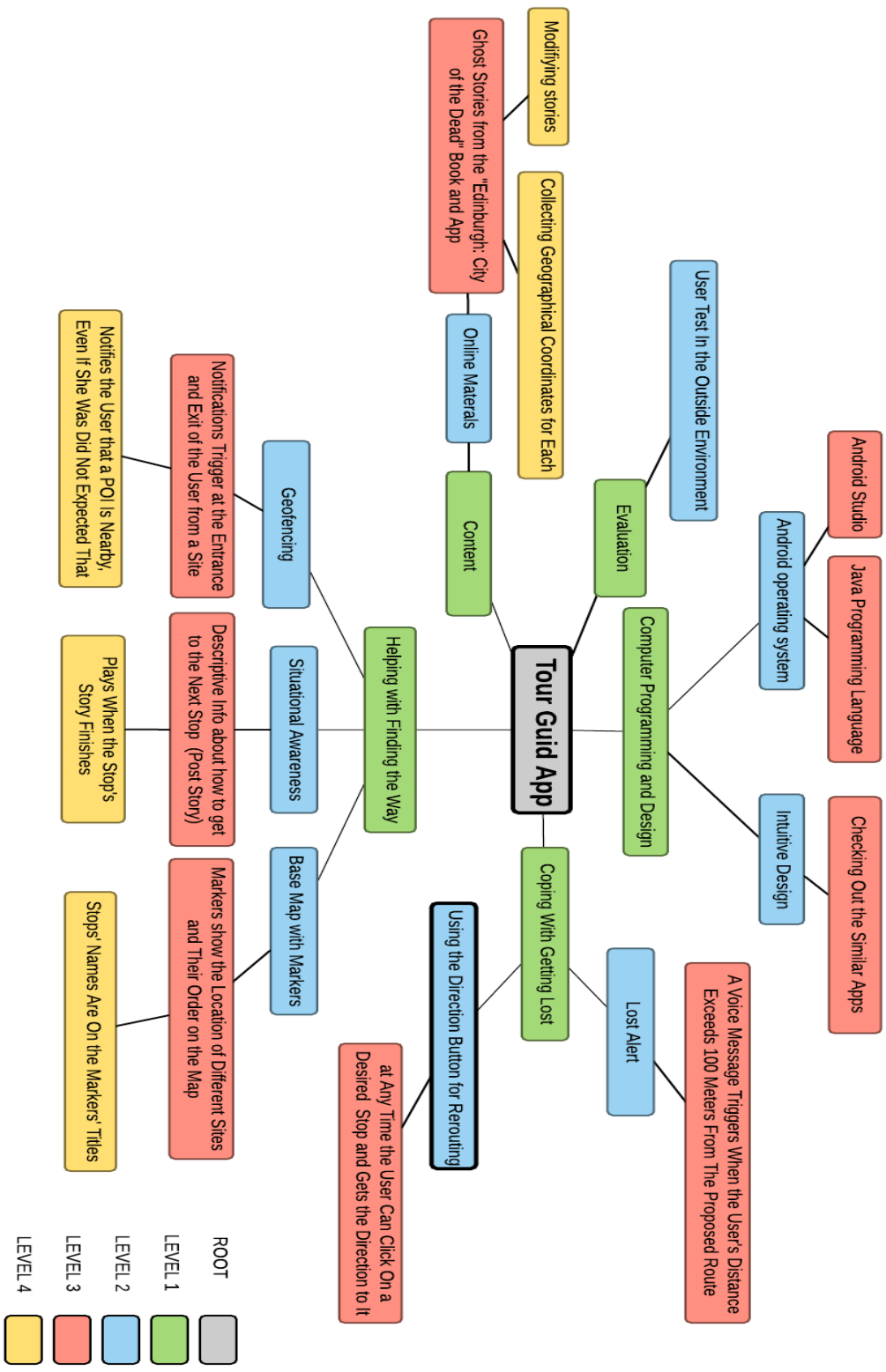


Figure 2. The mind map of the Tour Guide app

In this thesis, the following challenges are addressed: 1) Collection of the content, i.e., the stories to be told and how they connect with the geography that a user travels through; 2) Guiding the user to find his/her way through the tour; 3) Coping with the user's getting lost; 4) Programming the mobile application; and 5) Evaluating the app by testing it in the outside environment. Figure 2 shows the mind map of this project which is organized around the above five challenges.

The rest of the thesis is organized as follows. Other existing apps are summarized in section 3. The user interface and overview of the functionality of the app are discussed in section 4. Section 5 describes how the content was collected. Sections 6 and 7 discuss different techniques that are used to guide the user through the app in a ubiquitous way. Section 8 discusses different scenarios of going on a tour and section 9 explains how the app helps the user cope with getting lost. User testing, results and discussion are presented in sections 10, 11 and 12, respectively. Section 13 concludes the thesis and presents future work directions.

3 Other Existing Apps

Multiple walking tour apps have been developed in recent years, but none of them confront the storytelling in space and time in a way that the Tour Guide does. In particular, Tour Guide is a ubiquitous app with an intuitive design that tells situated stories of the city, helps the user find his/her ways to the stops and cope with getting lost.

In particular, the City of Dead (CityoftheDead, 2018) and the Rick Steve's Europe audio tour (Rick Steve's Europe, 2018) are two walking tour apps for Edinburgh. The City of Dead app, uses a Google map with markers on it showing the locations of different stops. However, it lacks additional features to help with finding the exact location of the stops or to cope with getting lost. The Rick Steve's Europe audio tour, narrates a series of stories related to the city and gives some SA information to help the user find his/her way, but it does not use a map. Hence, the user should rely only on how well he/she knows the city in order to find the places.

In addition, two other theses were done in 2014 (S1264598, 2014) and 2013 (Zhu, 2013) in the university of Edinburgh, school of geosciences, with the goal of building storytelling mobile app tours. However, they lack features to cope with getting lost; have no routing algorithm; do not support different usage scenarios; and also have some implementation issues regarding the use of geofencing techniques.

4 User Interface and Overview of the Functionality

The app is designed to have a visually appealing and intuitively obvious layout. I chose the design of the app based on the principle that when a user expresses that a design is intuitive, it is because s(he) has encountered a similar design before (Duret-Lutz, 2018). To this end, I got inspired by other well-known android walking tour apps including Rick Steve's Europe audio tour (Rick Steve's Europe, 2018), voice map (VoiceMapLtd, 2018b), haunted Edinburgh (CityoftheDead, 2018), walking tour and detour (Detour, 2018).

The app facilitates conveying all information by letting the user listen to them using Android's text to speech technology (AndroidDevelopers, 2018). This includes stories, notification, or general information about the app. The user is also able to read the contents should s(he) be willing to.

In this section, I briefly introduce the user interface of the app to familiarize the reader with the app's functionality. The app starts when the user clicks on its icon (Figure 3). The first page of the app contains three tabs, namely "Introduction", "Tour" and "History"[1], which are described below.

[1] The full content for this tab is available on the attachments in the technical report.

Figure 3. The Tour Guide app's icon

Introduction Tab.

This tab introduces the tour and demonstrates how to use the app (Figure 4). The introduction tab has the minimum necessary information for a user to start the tour without any additional help.

Tour Tab.

This tab lets the user choose his/her tour mode and starting stop (Figure 5). The user chooses to perform one of the following by pressing the associated buttons:

- Start the tour from the first site: The map with all the stops and the direction (from the current location) to the first stop will be shown (Figure 7).
- Start the tour from the closest site: An example is shown in Figure 8.
- Home mode: Since not all the users might be interested to start the tour right away, the user can view the stops on the map and listen to stories without physically going on the tour (Figure 9).
- Reset button: The reset button is to clear the user's walking history.

If the user presses any of the above three first buttons, a map will be shown. The map page has Google's base map with numeric markers. The markers show the location of each stops and their numbers guide the user about the order of the tour's stops. All markers are colored red at the beginning, which means none of the stops has been visited yet. When the user enters one of the stops, the color of its marker changes from red to green.

History Tab.

This tab gives more information about the history of the city from middle ages to present (Figure 6). The app uses geo-fencing techniques (Section 6) to give proper notifications to the user as s(he) enters or exits one of the sites. The user can choose to listen to the stories while at one of the sites, or at any other convenient time (Section 7.2). The story page is designed for listening to the story. Figure 10 is an example of the story page from the Lawnmarket and the Victoria street stop.

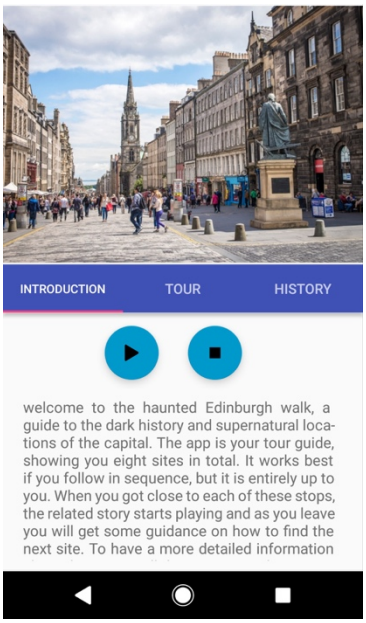


Figure 4: The introduction tab

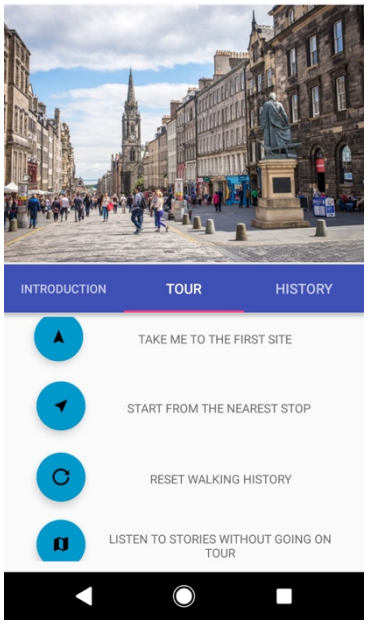


Figure 5: The tour tab

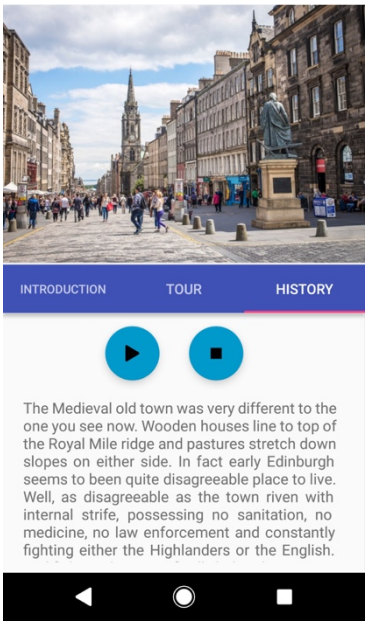


Figure 6: The history tab

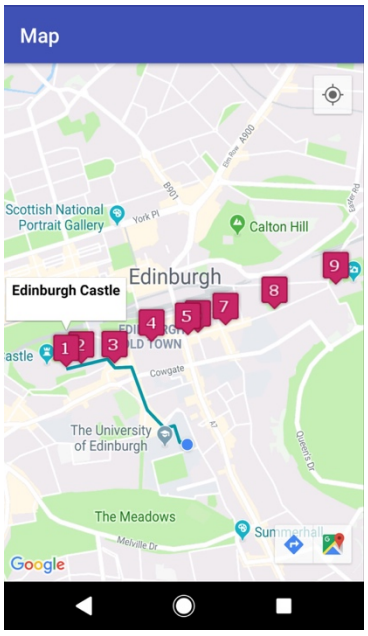


Figure 7: Direction from the current location to the first stop

Figure 8: Direction from the current location to the closest stop

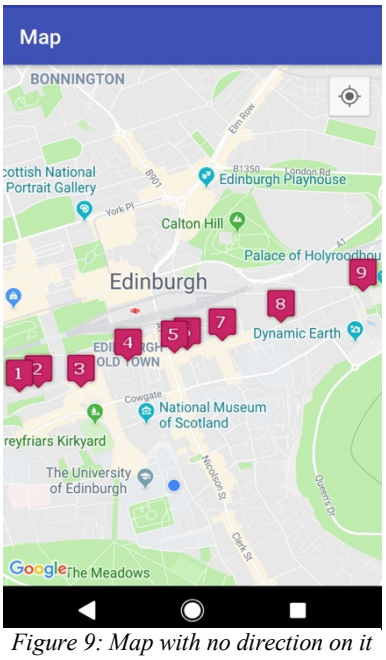


Figure 9: Map with no direction on it showing only the location of the stops

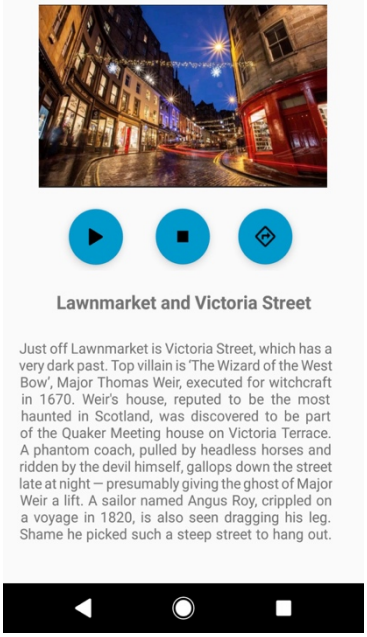


Figure 10: The Lawnmarket and Victoria street story page

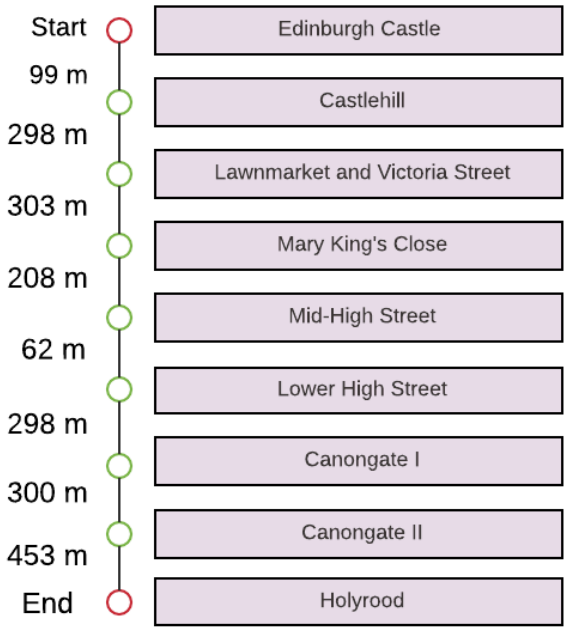


Figure 11. Names of the Tour Guide's stops and their distance from each other

5 Collection of the Content

Edinburgh has a reputation as one of the world's most paranormal cities in the world. The Tour Guide app uses Edinburgh's ghost tour in the famous Royal Mile as its case study. The content includes the geographical coordinates of each site as well as the stories. Among multiple online resources, I used the "Edinburgh: City of the Dead" book (Henderson, 2004) as it seemed most relevant to the project's purpose. I further used materials taken from some other walking tour apps for Edinburgh.[2]

The collected tour has nine sites. The stop's names and their distances to each other are shown in Figure 11. I collected the geographical coordinates of the stops using a GPS.

The project's story content has light hearted and narrative format. It also has a sequential format if the user follows the tour in order (Section 8.2). The story of each site has the following three components: 1) pre-story: information about the surrounding geography of the site; 2) (ghost) story: stories about the site; and 3) post-story: situational awareness direction information to the user (section 7.3). In the normal case of using the app, the full story of each site consists of concatenation of the three parts. Figure 12 shows an example of a full story for the "Canongate I" site. The app shows the pre-story only when the user is physically at the site, and the post-story if the SA is found to be useful for the user based on the next stop (Section 8.3).

[2] More details are discussed in the technical report.

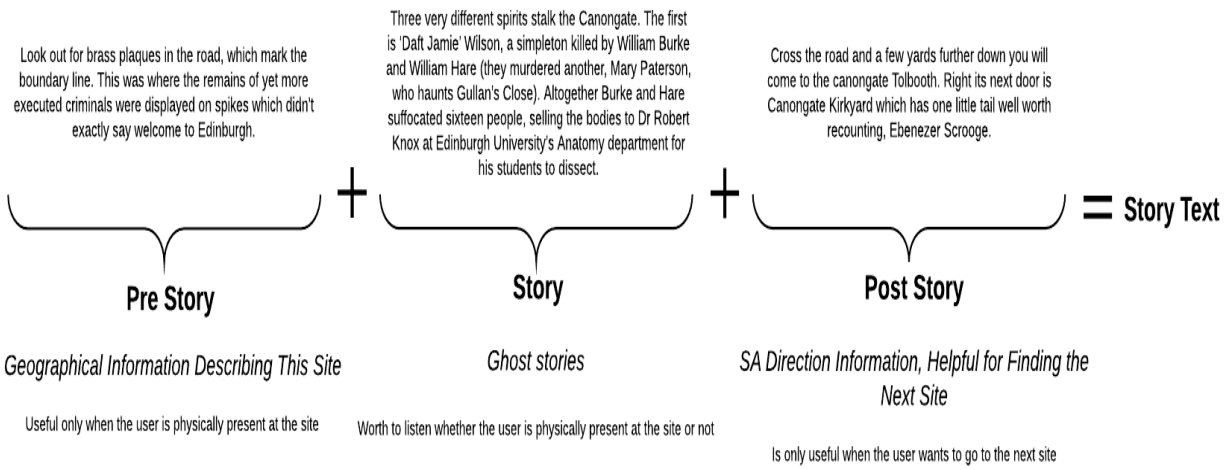


Figure 12. Story components

6 Geofencing

6.1 What Is Geofencing?

Geofencing is an LBS in which an app uses GPS, Wi-Fi or cellular data to trigger a pre-programmed action when a mobile device enters (Figure 13), dwells (Figure 14) or exits (Figure 15) a virtual perimeter (AndroidDevelopers, 2017). This virtual perimeter is known as a geofence and is set on a real geographic area (Megali, 2016). A circular geofence is specified by a latitude-longitude pair (center of geofence) and a radius. Typically, depending on how a geofence is configured, it activates some events on the mobile phone.

Geofencing has been used in numerous applications such as alerting users for shops that have certain sales, turning on and off the heating system of a house before its owners arrives or when they leaves (Leddy, 2016).

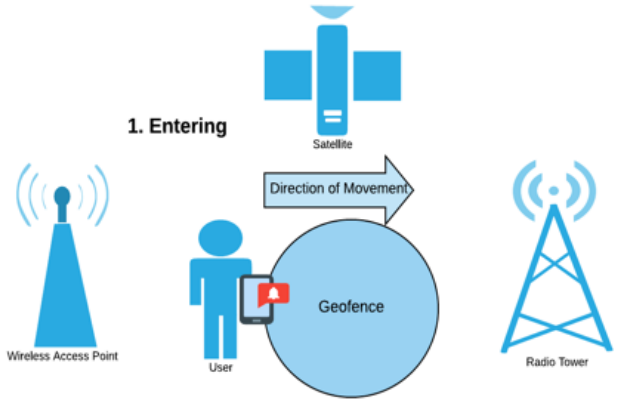


Figure 13. A user entering a geofence

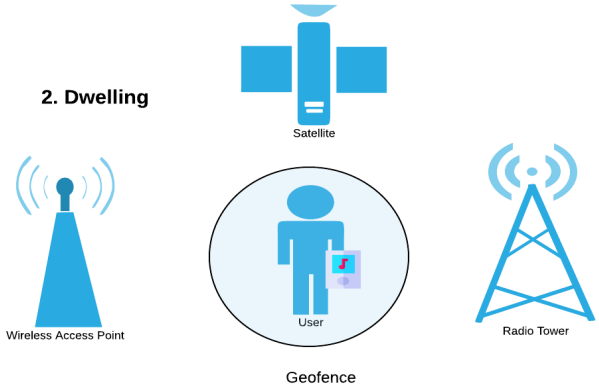


Figure 14. A user dwelling at a geofence for a certain amount of time

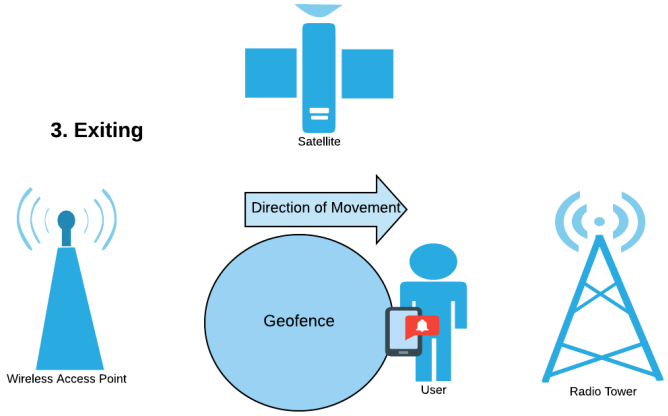


Figure 15. A user exiting a geofence

6.2 Geofences of the Tour Guide

The Tour Guide's geofences have circular shapes and are defined on the Areas of Interests (AOIs), where the tour's sites are located. Figure 16 shows the name and location of the nine geofences on the map.

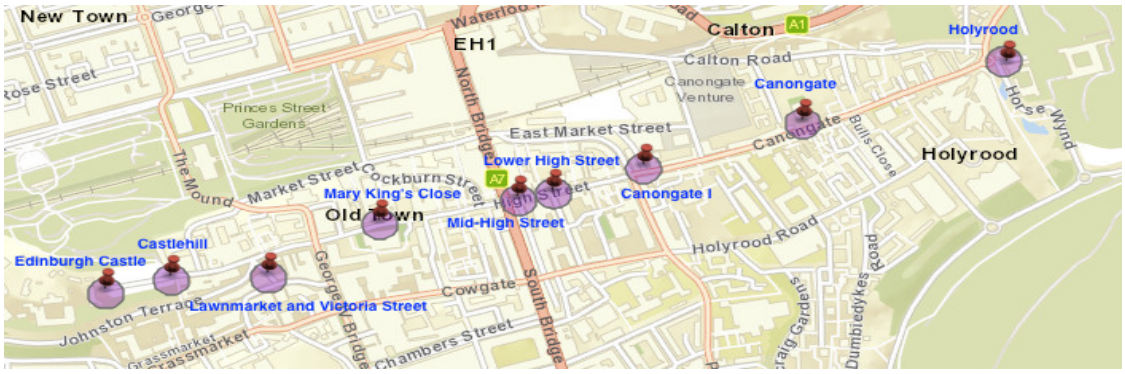


Figure 16. Geofences of the Tour Guide app on purple circles and their centers on red pins

The radius of the geofences should be set properly. A large radius makes it difficult for the user to find the site quickly and might lead to collision of nearby geofences. A small radius, on the other hand, makes the geofence

easily being missed by the user. I did field testing with multiple radius values[3] and found 35 meters to be reasonable for having effectively working geofences.

6.3 Tour Guide's Actions on Entering and Existing Geofences

The tour guide app performs some actions when the user enters or exits a geofence. In particular, when the user enters a geofence, an "Entrance Notification" is triggered on his/her phone, letting the user know which stop s(he) has entered. A voice message is released and lets him/her know that s(he) can click on the entrance notification to listen to the story. If the user presses the entrance notification, the app opens the story page and the story starts playing automatically. After the story reading finishes, the story page exits automatically. Alternatively, the user can press the back button or the direction button to exit the story page. In all cases, the map page will re-open and show a route to the next stop. It will also turn the seen stop's color to green (Figure 17).

As the user exits the geofence of the current stop, an "Exit Notification" pops-up on the screen and a voice message is released. After this notification the map gets updated and shows the direction to the next stop. Figure 18 and Figure 19 show flowcharts of entering and exiting a geofence, respectively.

It is notable that the GPS does not provide the exact location of the user and the user might sometimes move at the boundary of a geofence. Therefore, the geofencing implementations are known to give false/repeated alarms of enter and exit notifications. Dwelling times are usually used to overcome this problem (AndroidDevelopers, 2017). However, in the Guide Tour app, I found it best to only use the enter and exit notifications, but not that of dwelling. This is because we are interested in quick notifications, but dwelling times will prevent that. In order to make sure that the user does not get repeated notifications, the app keeps track of the last entered and exited geofences and gives exactly one notification for each event.

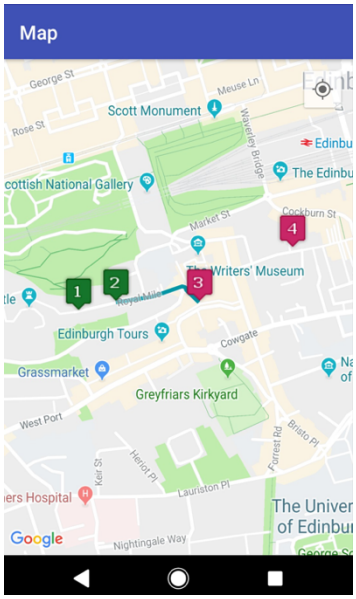


Figure 17. User currently finished listening to the story of the 2nd stop.

[3] I tested with radius of 15, 25, 35, 50 and 100.

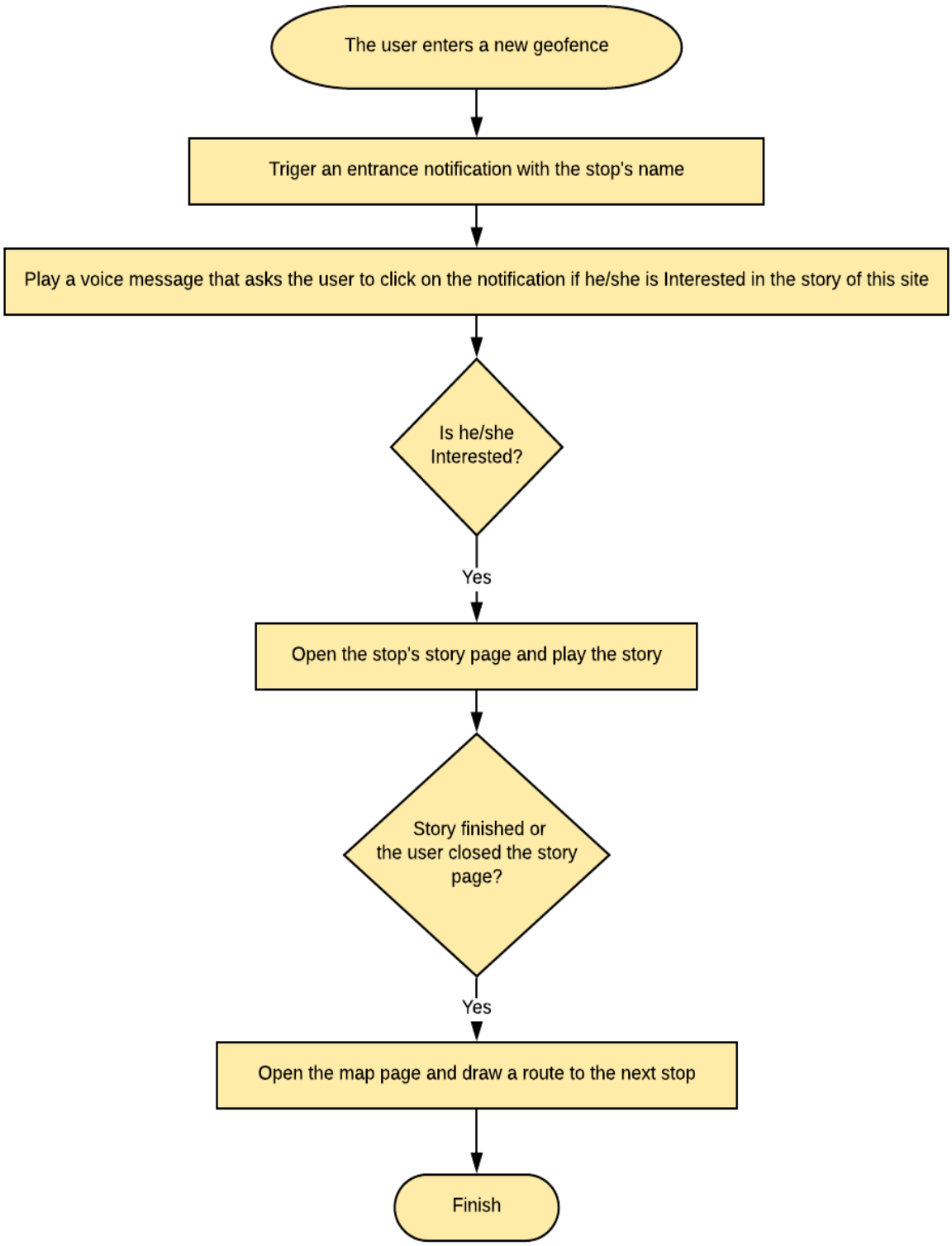


Figure 18. Flowchart for entering a geofence

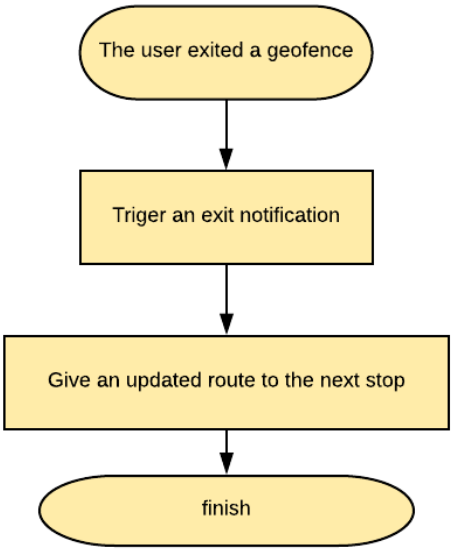


Figure 19. Flowchart for exiting a geofence

In order to make the app working ubiquitously, the enter and exit notifications will have minimum interfere with the usage of the app. In both cases, a notification will be shown on the phone (Figure 20 and Figure 21). The notification sound will be released only if the app is not already narrating a story. In the exit case, it is perfectly fine if the user does not click on the notification. However, for the enter notification, the user needs to press the notification for the story to start. This is the only case throughout the tour that the user has to touch the screen. This was inevitable because of two reasons: a) the app must consider user's safety. Hence, the user should not be distracted by the automatic start of a story; and b) it is usually more convenient for the user to find a nice spot before listening to the story.

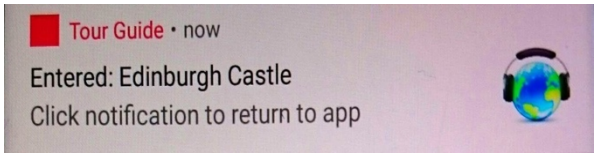


Figure 20. Entrance notification

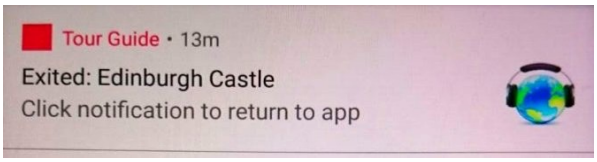


Figure 21. Exit notification

7 Other Approaches to Finding the Direction

One of the project's challenges is to help the user find his/her way through different stops and make sure s(he) figures out which site (e.g., street, building, close) each story is about. To this end, several techniques other than geofencing are used which are described below.

7.1 Google Map with Markers

The app has a Google base map with markers pinned on it showing the location of the stops.

7.2 Direction Buttons

The user can click on each of the markers on the map (Figure 22) to read the stop's name on the title (Figure 23), then click on the title to go to the stop's story page. Figure 24 shows the story page for the Mid-High street stop. The story page has three buttons:

 for playing the story.  for stopping it and  for getting the direction to that stop.

7.3 Situational Awareness Direction Information

To facilitate the process of finding the stops without the need to look at the map, some Situational Awareness direction information has been added at the end of the story for each site. The SA direction information is similar to how a person might give directions by using landmarks, colors and street names. It guides the user on how to get to the next stop, what to expect to see on the way and at the next stop. Figure 25 shows an example.

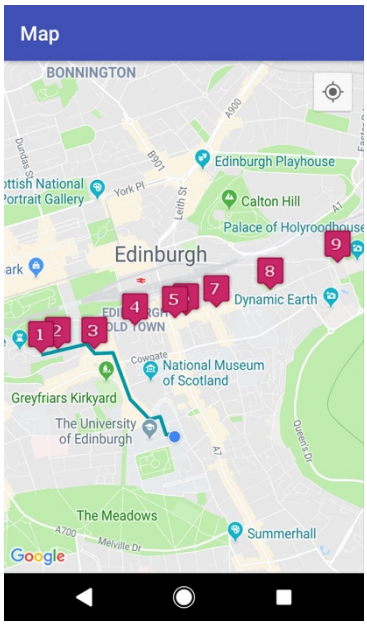


Figure 22. Map Page with a blue line showing the route to the 1st stop

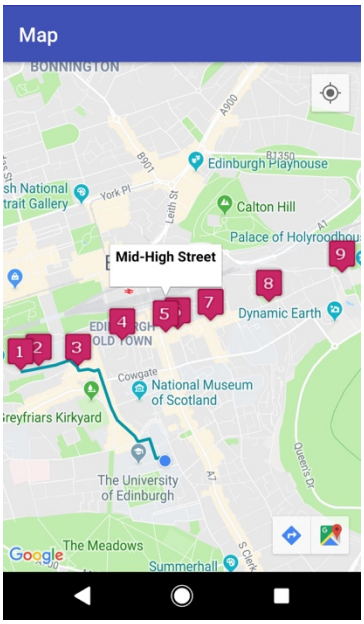


Figure 23. Map page with a selected marker

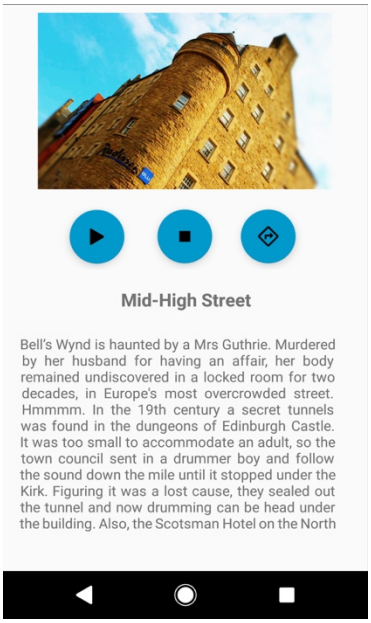


Figure 24. Story page of the selected marker (Mid-High Street)

7.4 Picture

For every stop, there is a picture from the site on the top of the story page that is served as a physical clue to the user. Figure 26 shows an example.

8 Tour Scenarios

The Tour Guide app has been developed in a way to support diverse usage behaviors of its users. In section 8.1, I discuss the order in which stops can be visited by the user. Then two example scenarios are provided in sections 8.2 and 8.3.

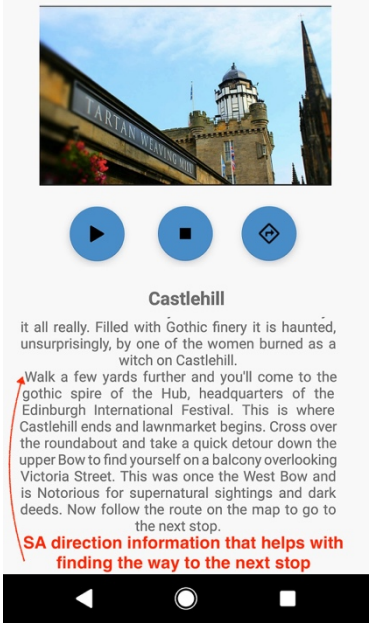


Figure 25. The SA information shown to the user in the Castlehill stop

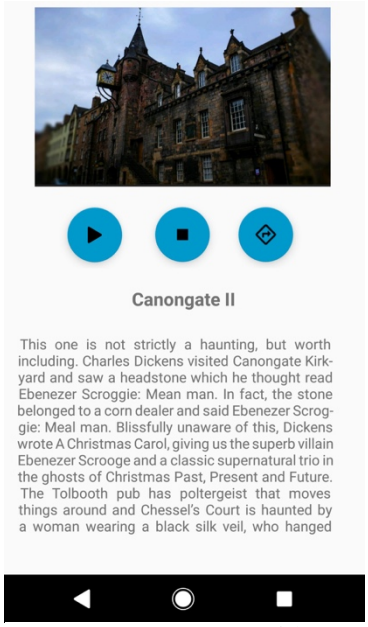


Figure 26. A picture of the Canongate at the top of the story page

8.1 Order of Visiting the Stops

From the beginning of the tour until the end, the app guides the user to go to next stops. At the beginning of the tour, the direction to the first or closest stop will be shown, depending on the user's choice. The next stop and the direction shown on the app get updated throughout the tour.

8.1.1 Automatically Suggested Next Stop

After the story of the current stop finishes, or the user exits the geofence of a stop, the direction to the next stop is shown to the user automatically. The app suggests the next unvisited stop in the natural order. For example, if the user has visited the stops in the order of 1, 2 and 3, the suggested stop will be 4. But if the user has gone in the order of 7, 1 and 5, the suggested stop will be 6. Finally, if the user gets to the last stop, but has not seen all the stops yet, the app suggests the first unvisited stop from the beginning. For example, if the user has gone in the order of 1, 5 and 9, the suggested stop will be 2.

8.1.2 Manually Selected Next Stop

As discussed in section 2 the user can manually choose a desired stop for visiting and getting the direction to it. This can be done at any time and will override the automatic suggestion.

8.2 First Scenario: Following the tour in the app's laid out order

The user prefers to follow the tour in the natural order (Figure 27). The user starts the tour from the first stop. The user follows the direction to the first stop and enters its geofence. The app gives a notification to the user and s(he) can choose to listen to the story. After the story is finished or the user exits the geofence, the direction on the map will be updated automatically. This process continues until the user visits all the stops.

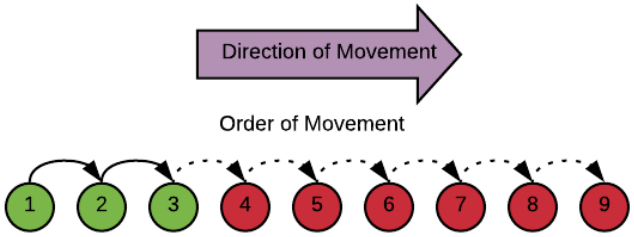


Figure 27. First scenario

8.3 Second scenario: Following an arbitrary order

Figure 28 shows a user that visits stop 1, then chooses to visit stop 4, 6 and 8. It is notable that if the user passes through one of the other stops other than the one that s(he) has selected, the app still shows a notification and the user can choose to listen to the story. However, the next stop will be the one s(he) has selected. The app will not suggest a new next stop, until the user enters the stop that s(he) has selected himself/herself.

For example, assume that the user selects to go to stop 4 after visiting the first stop. However, s(he) passes through stop 2. The app will give a notification and the user gets interested in that story and listens to it. After the story page is closed, the app will still show direction to stop 4 (instead of stop 3), as that has been the primary request of the user.

Note that as long as the requested stop is not the same as the next stop, the SA direction information will not be provided. For example, if the requested stop is 4 and the user passes through stop 2 and listens to its story, SA information will not be shown. This is because the user is going to stop 4 (not stop 3) and the app currently has SA information for going from stop 2 to stop 3, but not stop 4.

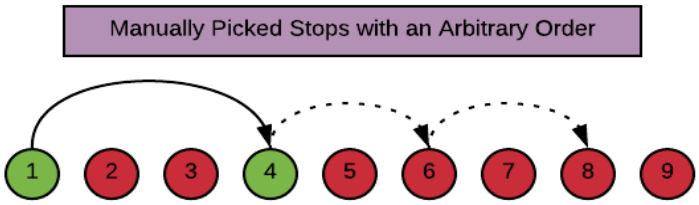


Figure 28. Second scenario

9 Coping with Getting Lost

The app is designed to help the user continue the tour when s(he) deviates from the suggested direction. The app automatically detects such cases (Section 9.1) and handles them (Section 9.2).

9.1 Deviation from the Direction

The app tracks the user's location and detects when s(he) deviates from the tour. This happens when the user gets lost or exits the tour temporarily for other reasons such as having a coffee. The app gets an update for location every $S=7$ seconds or when the user moves minimum of $M=10$ meters.

The app considers a user deviated from the tour when a) the map page is open, meaning that the user is supposedly looking for the next stop; and b) the distance between the user's current location to the suggested route on the map exceeds *C=70* meters (see Figure 29 for an example) for *K=3* consecutive times.[4] I tuned the parameters *S*, *M*, *C* and *K* on the field. The parameters were tuned considering three criteria:

- the app should not consume too much battery
- it should alert the user about deviation in a timely manner
- it should not give false alarms.

9.2 Recovering from Getting Lost

When the app detects for the first time that the user has deviated from the tour, it releases the following voice alert and updates the direction on the map based on his/her current location: "Wrong direction. Please check the updated map". Deviation from the tour might be because of reasons other than getting lost. For example, the user might want to buy a coffee on his/her path. As such, the app gives the wrong direction alert only once per destination. However, the user can always ask for an update on the direction by manually asking for direction to the desired stop.

Finally, one might argue that providing a continuously updating route, regardless of any deviation from the route (e.g., every 5 or 6 seconds), will better support the user as s(he) moves. However, this is not feasible for practical limitations.[5] This is similar to Google Maps where direction gets updated after the user deviates considerably from the suggested route. Figure 30 shows the flowchart of the mechanism used for coping with getting lost.

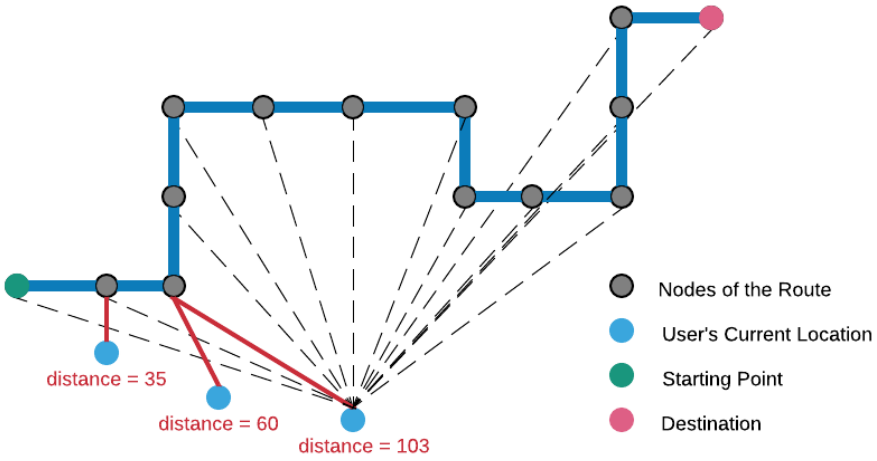


Figure 29. Example of user's distance to the suggested route

[4] In this project, the app always considers one route on the map, but the definition of deviation from the tour can be amended with two or more possible routes.
[5] I use Google's free API keys which provides a maximum number of queries both per second and per day. Please see Technical Report for more details.

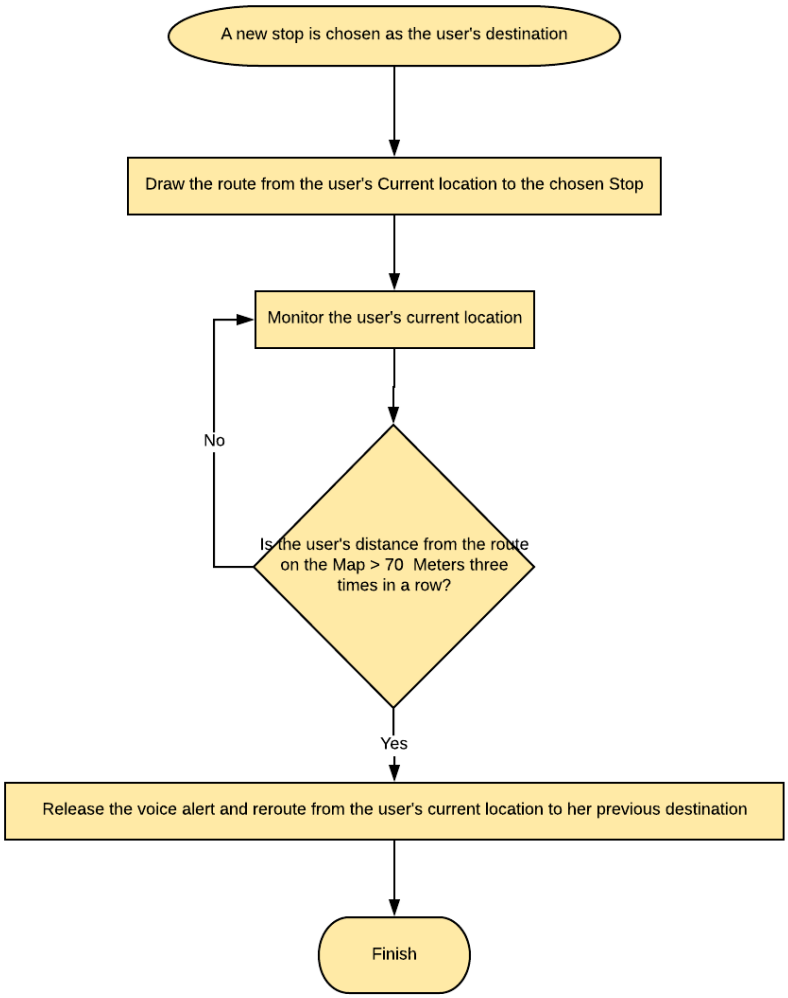


Figure 30. Flowchart for recovering from getting lost

10 User Testing

The user testing is designed to assess how successful the Tour Guide app is in answering to the project's research questions (Section 2). Testing was done with eight users through July 2018. Before each of the users start the tour by their own, they were given a one-page document followed by a questionnaire (Please see Appendix II). The document contained information about the purpose behind creation of the app, how the user can work with the app, ethics reviews and safety tips.

This questionnaire contained questions which were graded on the Likert scale, from strongly disagree (negative) to strongly agree (positive). These questions evaluated the strengths and weaknesses of the app in its five main aspects (questions 6-13): 1) content of the app; 2) its design; 3) the geofencing technique; 4) TTS; and 5) navigation. In addition, some other multiple-choice questions were designed to assess the app's functionality. The users were encouraged to add their own comments (questions 14-17).

Testing took place in two stages. At the first stage, two users tested the app and expressed their comments on the app. Fortunately, the app did not have any serious issues, but some small improvements suggested by the

users were done to the app. Then the other six users performed the testing. No further improvements were done based on the comments of the six users, but those suggestions provide directions for future work. The users took the author's Xperia XZ (release date: Sep 2016) device and went on the tour by their own. The age-range of the users was 23 years old to 34 years old. The users were both male (4) and female (4) and their occupation was mostly student.

11 Results

The questions are divided into 9 categories and their results are reported accordingly in the following subsections.

11.1 Mobile Tour versus Human Tour

The results show that for all of the users, it was their first time experiencing a walking tour mobile app; however, 25% of them had a human walking tour experience before. 100% of the users agreed that using the app was fun. They mostly preferred a mobile tour over a human tour (Figure 31).

11.2 Content

All users expressed that they have strongly enjoyed the content of the app (Figure 32).

11.3 App design

All users seemed to have enjoyed the app design, by 75% selecting strongly agree and 25% selecting agree to the question (Figure 33).

11.4 Text to Speech

The text to speech has worked properly, by 62.5% strongly agreeing with the statement, 25% agreeing and only 12.5% being neutral (Figure 34).

11.5 Geofencing

Figure 35 and Figure 36 show that the geofencing and their notifications have worked well most of the time, with 12.5% or 25% of the users having some concerns (neutral).

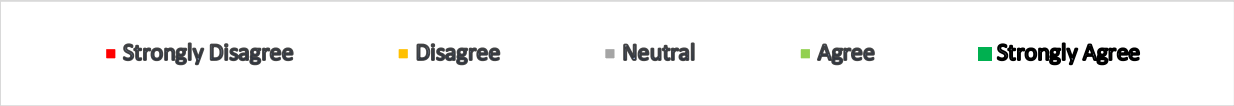
11.6 Navigation

The users mostly found it easy to navigate with the app and find their ways to next stops., except for one user whose testing was done during rain showers (Figure 37). The users were also asked which of the SA information or the route on the map helped them more in finding the way (Figure 38). The result showed that they mostly relied on the directions on the map (75%), but some preferred the SA information (25%).

11.7 Strongest and Weakest Aspect of the App

The users were asked what they like most about the app? The users' answers from most repeated to least repeated are:

- Look and Feel (4 users)
- Content (4 users)
- Functionality (2 users)
- Speed (2 users)
- Navigation (1 user)



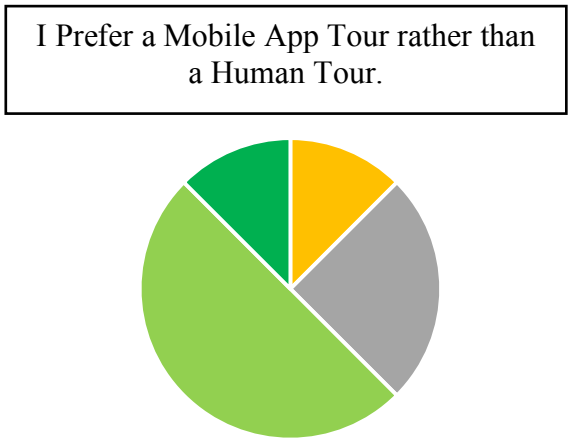


Figure 31. User's opinion on whether they prefer a mobile tour over a human tour

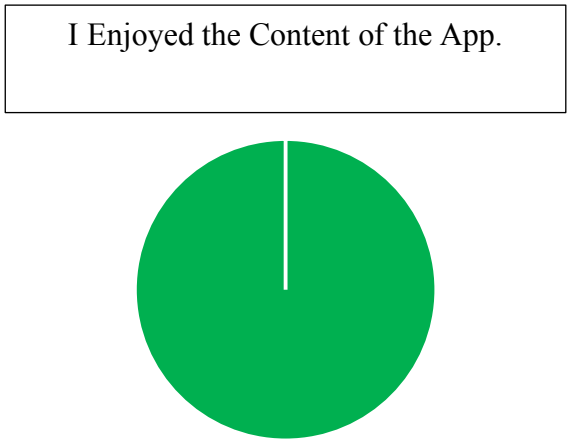


Figure 32. User's opinion on the content of the app

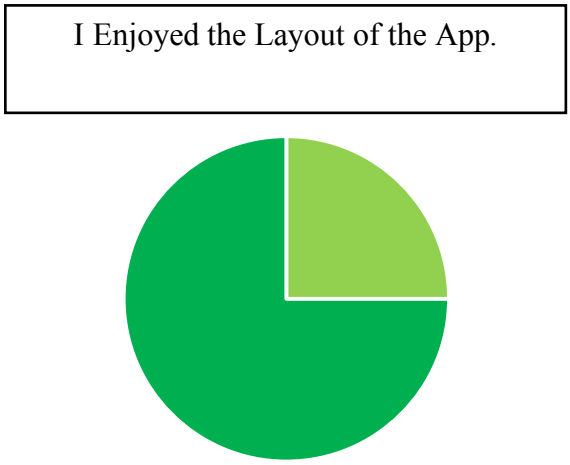


Figure 33. User's opinion on the design of the app

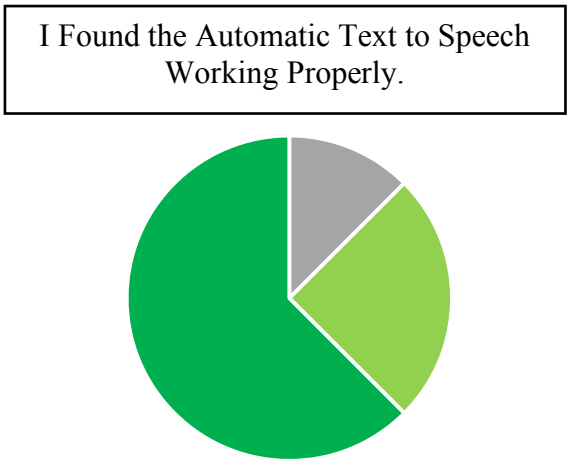
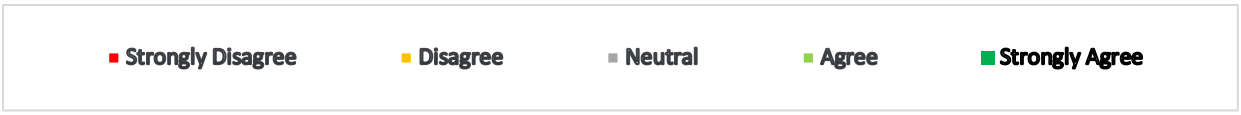


Figure 34. User's opinion on the TTS engine



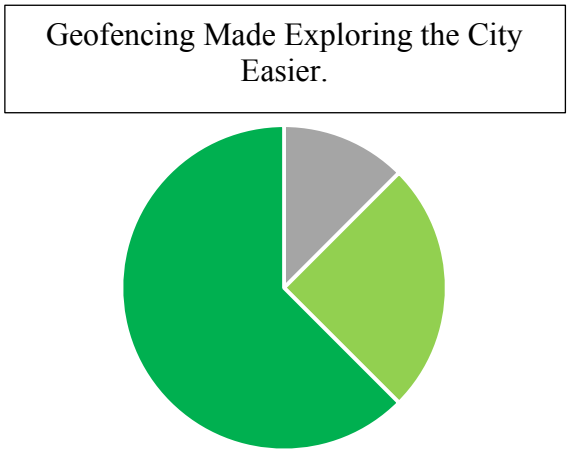


Figure 35. User's opinion on the functionality of the geofencing technique

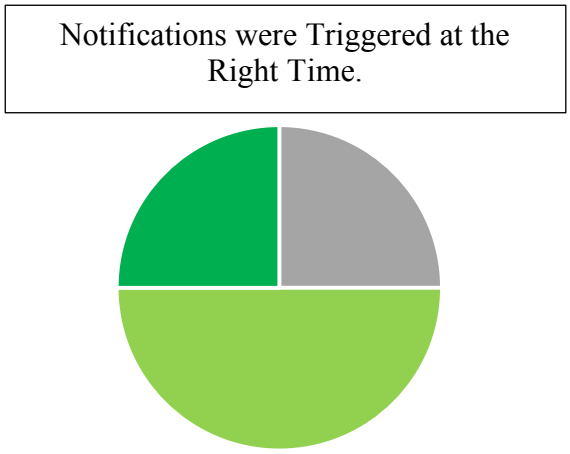


Figure 36. User's opinion on the accuracy of the geofence

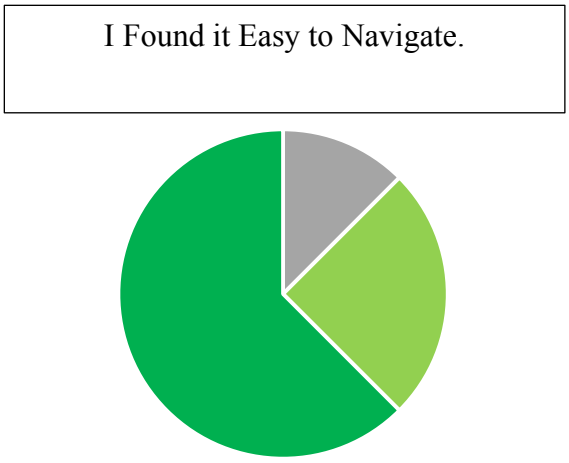


Figure 37. User's opinion on whether it was easy to navigate using the app

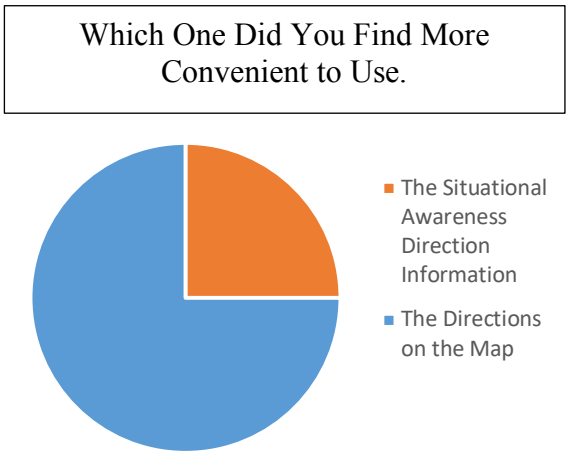


Figure 38. Users' opinion on comparison between SA direction information and the route on the map

Then the users were asked what was the biggest issues with the app? Answers from most repeated to least repeated are:

- False alarm for getting lost (2 users): reported by the first two testers and resolved immediately by tuning the parameter *C* for coping with getting lost (see Section 9 and the technical report).
- Lack of pausing button (2 users): left to future work as there was no plan for additional user testing.
- The app's inability to work in existence of thick clouds (1 user): Unfortunately, nothing much can be done about this issue, as it is mostly a hardware issue, but other or more modern mobile brands might have a better built-in GPS.

11.8 Requested Improvements

The users were asked about their desired features to be added to the app? Answers from most requested to least repeated are:

- Acknowledgement at the end of the tour (2 users): reported by the first two users and resolved.
- Use of recorded sound (especially Scottish accent) instead of TTS (2 users): left to future work.
- Adding the pause button (2 users): left to future work
- Paragraph breaks at the stories to make it easier for reading (1 user): one of the first two users mentioned this request which was resolved.
- More descriptive / visual SA direction information (1 user): left to future work.

11.9 Overall App Rating

The overall app rating was very high, by six users (75%) giving a score of five and two users (25%) giving a score of four (Figure 39).

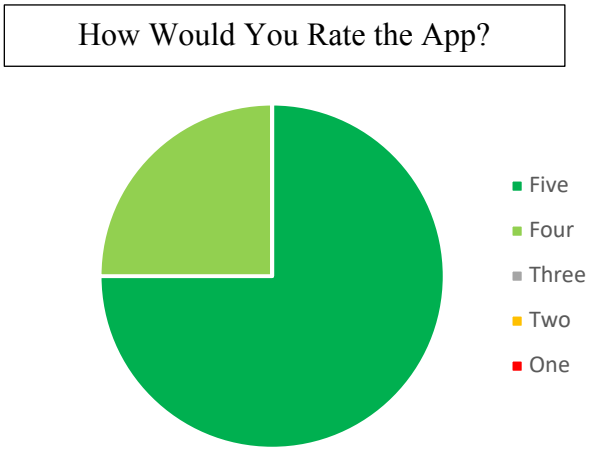


Figure 39. User's overall rating of the app

12 Discussion

The results of the project were driven by the four main research questions:

- Can we use smartphone technology to guide us through the streets and tells us situated stories that take into account where we have just been, and where we are? In other word, can we have streetstorytell instead of streetview? (Mackaness, 2012)
- Is the android the right programming environment for creating the app?
- How well the app can perform the tour, versus a traditional human tour?

In answer to the first question the Tour Guide app was developed, and user testing was performed. The users gave an average rate of 4.75 (out of 5) to the app's overall rating. This high rate of the users' satisfaction answered the first research question positively and suggests that this project was successful in building a sample of such tours using a smartphone technology.

In answer to the second question, it should be said that Android holds the single biggest market share worldwide at 85% (InternationalDataCorporation, 2018). It has powerful APIs, excellent documentation, a thriving developer community, and no development or distribution cost (Hassan, 2008). The high rate of satisfaction from different features that Android provides for using in the app including geofencing technique, navigation, TTS, layout design, and look-and-feel confirm that the use of Android for the Tour Guide app satisfies the users' expectations. As such, Android can be considered as an ideal platform for this project.

In answer to the third question, user testing showed that most users prefer to go on a mobile app tour. This result suggests that the Tour Guide app as an example in the walking tour apps, performed the tour in a way that the users are happy to have a mobile walking tour instead of human tour. However, I think this claim needs further study because six users did not have any human tour before, so they had no previous experience to compare it with this one and their opinion may not correctly reflect the opinion of the society. The two other users who had the human tour experience before, answered to this question very differently. One of them strongly prefers a mobile tour, while the other one was disagreeing.

13 Conclusion and Future Work

Many researches have been conducted to integrate GPS positioning and multimedia functions on the smartphones to provide location based tour guide services (Lu and Arikawa, 2013). The mobile tours are typically least expensive than the human tours; they can be used at any time of the day; and for many times.

The Tour Guide android app proved an effective and novel method for creating a location based storytelling tour. This app helps its users find their way to the sites easily; narrates them fun stories at the right time; suggests them the next stop to visit while supporting their different choices; and helps them to get back to the tour's route with minimum effort after they get lost. The application of this app is not limited to the tourists, as some the users whom had been lived in the city for many years enjoyed the app, since they could explore the city in a different way.

While the app was well able to answer the project's research questions, future work can certainly improve it more from its current situation. Some of the future works were suggested by the users in the section 10.8. Additional further work includes:

1) Adding recommended personalized tours to the app. This feature will enable the app to recommend a personalized tour based on the tourist's personal interests, social backgrounds and transportation type by more complex routing algorithms (Anacleto et al., 2014, Owaied et al., 2011).
2) Using a combination of multimedia and Virtual Reality (VR) for effective storytelling, especially for sites which has turned to ruins to show them in their early forms.
3) Adding more tours to the tour to cover more cities, also satisfies users with different interests. For example, tours about: museums, galleries, statues, blue plaques, famous buildings, famous restaurant and food for different famous cities in the world.
4) Adding voice recognition to the app to give the user the possibility of putting his/her phone in his/her backpack and interacting with the app by voice.

14 References

ANACLETO, R., FIGUEIREDO, L., ALMEIDA, A. & NOVAIS, P. J. J. O. N. A. C. A. 2014. Mobile application to provide personalized sightseeing tours. 41**,** 56-64.

ANDROIDDEVELOPERS. 2017. *Creating and Monitoring Geofences* [Online]. AndroidDevelopers. Available: https://developer.android.com/training/location/geofencing.html [Accessed].

ANDROIDDEVELOPERS. 2018. *TextToSpeech* [Online]. Available: https://developer.android.com/reference/android/speech/tts/TextToSpeech [Accessed].

CITYOFTHEDEAD. 2018. *Haunted Edinburgh App | City of the Dead Tours* [Online]. Available: https://www.cityofthedeadtours.com/haunted-edinburgh-app/ [Accessed].

CLARK, K. Stories everywhere. Where 2.0''conference, 2011. 19-21.04.

DETOUR. 2018. *Detour - Guided Walking Tours* [Online]. Detour. Available: https://www.detour.com [Accessed].

DICKINSON, J. E., GHALI, K., CHERRETT, T., SPEED, C., DAVIES, N. & NORGATE, S. J. C. I. I. T. 2014. Tourism and the smartphone app: Capabilities, emerging practice and scope in the travel domain. 17**,** 84-101.

DURET-LUTZ, A. 2018. *What is Intuitive Design?* [Online]. The International Design Foundation. Available: https://www.interaction-design.org/literature/topics/intuitive-design [Accessed].

FESTA, J. 2016. Old-school guided tours transformed by new technology. *USA Today*.

HASSAN, Z. S. Ubiquitous computing and android. Digital Information Management, 2008. ICDIM 2008. Third International Conference on, 2008. IEEE, 166-171.

HENDERSON, J. A. 2004. *Edinburgh City of the Dead*.

INTERNATIONALDATACORPORATION. 2018. *IDC: Smartphone OS Market Share* [Online]. Available: https://www.idc.com/promo/smartphone-market-share [Accessed].

KANG, K., JWA, J. & PARK, S. E. J. I. J. O. A. E. R. 2017. Smart Audio Tour Guide System using TTS. 12**,** 9846-9852.

LEDDY, P. 2016. *7 Things About Geofencing You'll Kick Yourself for not Knowing* [Online]. PulsateHQ. Available: http://academy.pulsatehq.com/7-things-about-geofencing [Accessed].

LU, M. & ARIKAWA, M. 2013. Map-based storytelling tool for real-world walking tour. *Progress in location-based services*. Springer.

MACKANESS, W. 2012. *Story telling in space and time* [Online]. Available: https://www.geos.ed.ac.uk/to-auth/mscdb/see_diss_details.html?q_diss_id=1511 [Accessed].

MEGALI, T. 2016. *How to Work With Geofences on Android* [Online]. @tutsplus. Available: https://code.tutsplus.com/tutorials/how-to-work-with-geofences-on-android--cms-26639 [Accessed].

OWAIED, H., FARHAN, H., AL-HAWAMDEH, N. & AL-OKIALY, N. J. J. O. A. S. 2011. A model for intelligent tourism guide system. 11**,** 342-347.

RICK STEVE'S EUROPE, I. 2018. *Audio Tours for Europe by Rick Steves* [Online]. Available: https://www.ricksteves.com/watch-read-listen/audio/audio-tours [Accessed].

S1264598. 2014. *Storytelling in Space and Time.* Master of Science, University of Edinburgh.

SUI, D. & GOODCHILD, M. J. I. J. O. G. I. S. 2011. The convergence of GIS and social media: challenges for GIScience. 25**,** 1737-1748.

VOICEMAPLTD. 2018a. *Immersive GPS audio tours » VoiceMap* [Online]. Available: https://voicemap.me/ [Accessed].

VOICEMAPLTD. 2018b. *Walking Tour App » VoiceMap* [Online]. Available: https://voicemap.me/walking-tour-app [Accessed].

ZHU, Q. 2013. *Story Telling In Space and Time (An Android Application for Ghost Tour in Edinburgh using Smart Phones).* Master of Science, University of edinburgh.

Part II

Technical Report

Table of Contents

Table of Figures

1 Introduction

This technical report is written to describe the steps that were taken to create the Tour Guide app, also where possible explains the reason why a specific design choice was made. The main goal is to let researchers who are interested in this project explore its details and be able to expand it in future. In particular, this technical report describes: a) how to set up the necessary development environment; b) which parts of the code are taken from the GitHub (basic geofencing capabilities, base map and routing); c) how the full working Tour Guide app is implemented to address the project's challenges (collection of content, guiding the user to find his/her way, coping with the user's getting lost and evaluating the app by testing it in outside environment); and d) how the user can access the code and extend it.

2 Setting Up App Development Environment

Any app developments need a Software Development Kit (SDK) for creating the app. It is also convenient to use an Integrated Development Environment (IDE) to help with coding, spell checking, reporting warnings/errors, developing UI, etc (Joey, 2010). In Tour Guide app, the used programming language is Java, the code is developed based on Android operating system, and it is run on Android Platform. The choice of Android is because of its ubiquity of users worldwide and its wide range of APIs in the Google Play services that enable users to build high quality apps in relatively short time (AndroidDevelopers, 2018e).

In the Tour Guide project, I found the following APIs extremely helpful: Google Location and Activity Recognition (Location), Google Maps[1] and other well-documented libraries for routing (Section 3.1). These APIs add location awareness to the app which is useful for creating and monitoring geofences, automated location tracking and routing.

The code was developed using Android Studio IDE. Android Studio is the only and official IDE for Android, it comes with Android SDK and it offers features that enhance app building productivity (AndroidDevelopers, 2018f). Hence, it is an ideal IDE for this project. Some of these tools are: Intelligent code editor, which helps with code completion; and the visual layout editor that makes it easy for users to create complex layout using constraint layout (AndroidDevelopers, 2018a).

The workflow to develop an app for android is conceptually the same as other app platforms and it contains: 1) setting up the workspace; 2) writing the app; 3) building and running; 4) debugging and testing; and 5) publishing. Figure 1 shows this process on a diagram (Developers, 2018). These steps are explained specifically for this project.

2.1 Setting up the workspace

At the first step, I downloaded the Android studio installation file from its official website[2] and installed it on my computer. Fortunately, Android-Play-Location[3] provides a sample code for creating and monitoring geofences. I downloaded the code from GitHub[4] and imported the "geofencing" project into my working space. I extended the code based on the Tour Guide's requirements. I further used other codes from GitHub

[1] https://developers.google.com/android/guides/setup
[2] https://developer.android.com/studio/
[3] https://github.com/googlesamples/android-play-location
[4] https://github.com/googlesamples/android-play-location/tree/master/Geofencing

(e.g., code for creating a Google map) and created and developed other activities which I merged with the main project.

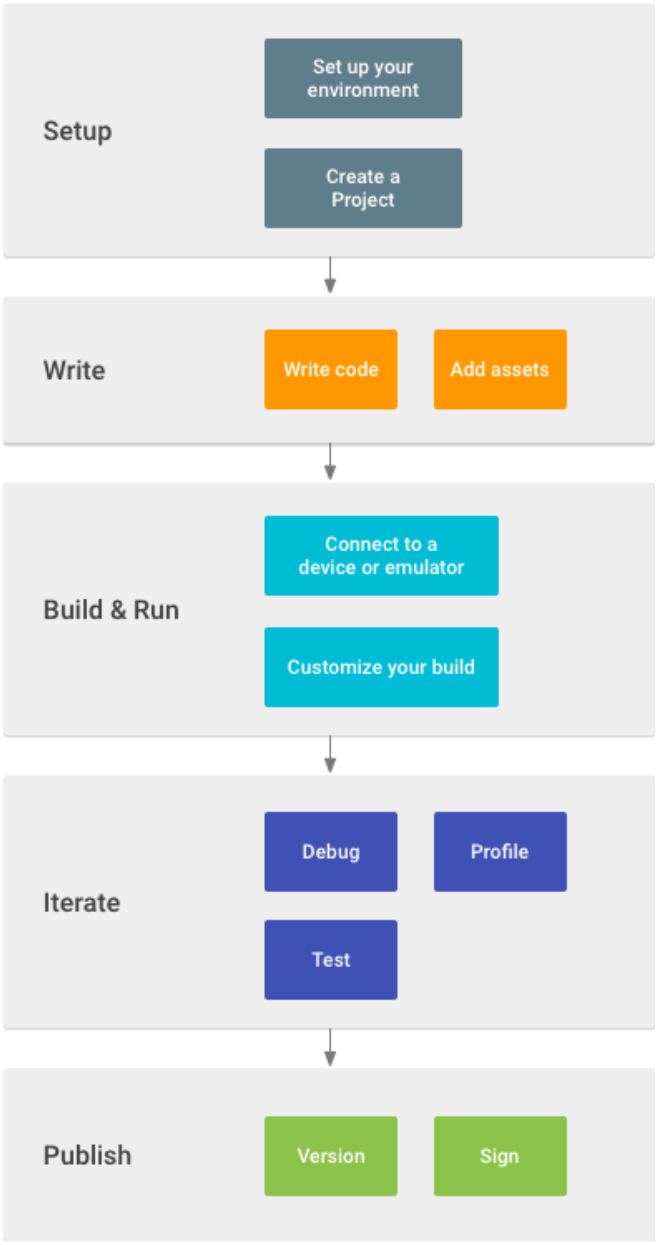


Figure 1. App development workflow

2.2 Writing the App

The app's code was developed based on the required functionalities. As discussed in step 1, I started with a simple geofencing code downloaded from GitHub and extended it to the full working project.

2.3 Building and running

I used Android studio's Gradle for building and running the application. I ran the code on my Xperia XZ mobile device (release date: September 2016).

2.4 Debugging and Testing

I performed most of the debugging in the home's neighborhood (located near King's Building) and the school of Geosciences. When the app seemed to work fine with a few Geofences, I performed more debugging and testing at the actual Ghost Tour. Finally, when the app was at a stage that could be successfully used to perform a full Ghost Tour, I asked eight human subjects to perform the testing (Section 9).

2.5 Publishing the Code

The code is freely available at GitHub[5] for academic use.

3 Structure of the Project

The Android studio has a default view for displaying the project's files, named "Android view". This view shows key source files of the project and has two main modules: App and Gradle Scripts. These modules are described below for the Tour Guide app.

3.1 Gradle Script Module

Gradle is a built system that uses many tools and processes to convert the project into an Android Application Package (APK) (Montegriffo, 2018). This APK then can be used by the Android operating system for installation of the mobile app. This module makes it possible to configure some aspects of build. In this project the only build configuration I made was adding some build dependencies to the gradle (module:app), so I only discuss them (AndroidDevelopers, 2018c). Figure 2 shows the build.grade xml file of Tour Guide app.

Gradle:

Line 4: I compiled the Tour Guide app against the compileSdkVersion 26. This is the latest API level and enabled me to use the most up-to-date API features in my app.

Line 9: The minSdkVersion for this app is 16, which specifies the minimum required API level for a device so that it can run the app.

line 10: While I was developing the app, I was testing it on my Sony Xperia XZ device with an API level 26 (Android 8). So, I declared the targetSdkVersion to 26 for this app.

Line 24: This line adds the ConstraintLayout library as a dependency, which enables building complex UIs more easily.

[5] https://github.com/LalehMoussavi/storyTelling

Line 25: This library provides support for static vector graphics in the drawable folder.

Line 32: The library provides support for various material design components, e.g. FloatingActionButtons and Tabs.

Lines 33 – 35: These lines add Google Play Services available to the project. The two available services are Google Location and Activity Recognition (line 31), and Google Maps

Line 36. This library was added to the project to be used in routing.

```
1.  apply plugin: 'com.android.application'
2.
3.  android {
4.      compileSdkVersion 26
5.      buildToolsVersion '27.0.3'
6.
7.      defaultConfig {
8.          applicationId "com.google.android.gms.location.sample.geofencing"
9.          minSdkVersion 16
10.         targetSdkVersion 26
11.         versionCode 1
12.         versionName "1.0"
13.         vectorDrawables.useSupportLibrary = true
14.     }
15.     buildTypes {
16.         release {
17.             minifyEnabled false
18.             proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-
    rules.pro'
19.         }
20.     }
21. }
22.
23. dependencies {
24.     implementation 'com.android.support.constraint:constraint-layout:1.0.2'
25.     implementation 'com.android.support:support-vector-drawable:26.1.0'
26.     androidTestCompile('com.android.support.test.espresso:espresso-core:2.2.2', {
27.         exclude group: 'com.android.support', module: 'support-annotations'
28.     })
29.     compile 'com.android.support:appcompat-v7:26.1.0'
30.     testCompile 'junit:junit:4.12'
31.
32.     compile 'com.android.support:design:26.1.0'
33.     compile 'com.google.android.gms:play-services-location:11.0.0'
34.     compile 'com.google.android.gms:play-services-maps:11.0.0'
35.     compile 'com.google.android.gms:play-services:11.2.0'
36.     compile 'com.github.jd-alexander:library:1.1.0'
37.     compile 'com.google.maps:google-maps-services:0.1.20'
38.     compile 'com.mapzen.android:speakerbox:1.4.1'
39. }
```

Figure 2. Build.Grade XML file

3.2 App Module:

Within the App module there are four directories: manifests, java, assets, and res. The structure of these directories is discussed in this section. The Tour Guide app is developed around four activities: Main Activity, LandingPage Activity, TourGuideMap Activity and Story Activity. The manifests and assets directories contain files that are relevant to all activities, but the java and res directories contain files that are specific to only a subset of activities.

3.2.1 Manifests Folder

Purpose:

Every app project has an AndroidManifest.xml file (Figure 3), which describes essential information about the app. For example, it has information about Google play and Android operating system (AndroidDevelopers, 2018b).

Lines 1 - 2 specifies the app's package name, which is the same as the geofencing sample code because I start building the code on its base code.

Line 4 specifies that the app requests to access precise location of the user, using the GPS.

Line 5 declares that the app wants to keep processor from sleeping or screen from dimming.
Lines 7 -13 specifies the app's name and icon when installed on a device.

Lines 13 -15 adds the Google Play services version to the app.

Lines 18 – 20 adds the API key to the app. In order to get the API key, I did the following: In the Google Cloud Platform Council,[6] I created a new project and got an API key to access the Google servers. Then amongst all available maps APIs and services, I enabled the Maps SDK for Android and Directions API for the API key.

Lines 22 – 57 declare the components of the app, which include activities, services, broadcast receivers, and content providers.

Lines 22 - 30 means that the MainActivity is the entry point of the application, i.e. when the app is launched this activity is created.

```
1.  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
2.      package="com.google.android.gms.location.sample.geofencing">
3.
4.      <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
5.      <uses-permission android:name="android.permission.WAKE_LOCK" />
6.
7.      <application
8.          android:allowBackup="true"
9.          android:icon="@drawable/earth"
```

[6] https://console.cloud.google.com/

```
10.         android:label="@string/app_name"
11.         android:supportsRtl="true"
12.         android:theme="@style/Theme.Base">
13.         <meta-data
14.             android:name="com.google.android.gms.version"
15.             android:value="@integer/google_play_services_version" />
16.
17.         <!--             The API key for Google Maps-based APIs. -->
18.         <meta-data
19.             android:name="com.google.android.geo.API_KEY"
20.             android:value="AIzaSyCeQxKit_dVaE5nn74A1yA7xYlSraEOPUk" />
21.
22.         <activity
23.             android:name=".MainActivity"
24.             android:label="@string/app_name">
25.             <intent-filter>
26.                 <action android:name="android.intent.action.MAIN" />
27.
28.                 <category android:name="android.intent.category.LAUNCHER" />
29.             </intent-filter>
30.         </activity>
31.
32.         <receiver
33.             android:name=".GeofenceBroadcastReceiver"
34.             android:enabled="true"
35.             android:exported="true" />
36.
37.         <service
38.             android:name=".GeofenceTransitionsJobIntentService"
39.             android:exported="true"
40.             android:permission="android.permission.BIND_JOB_SERVICE" />
41.
42.         <activity
43.             android:name=".Story"
44.             android:label="@string/title_activity_scrolling"
45.             android:theme="@style/Theme.Base.NoActionBar" />
46.         <activity
47.             android:name=".Main2Activity"
48.             android:label="@string/title_activity_main2" />
49.         <activity
50.             android:name=".MapsActivityCurrentPlace"
51.             android:label="@string/title_activity_maps"
52.             android:theme="@style/GMAppTheme" />
53.         <activity
54.             android:name=".Introduction"
55.             android:label="@string/title_activity_introduction"
56.             android:theme="@style/Theme.Base.NoActionBar"></activity>
57.     </application>
58.
59. </manifest>
```

Figure 3. Android Manifest XML file

3.2.2 Java Folder

Contains the Java source code files, separated by package names. The Java codes are described in the following sections.

3.2.3 Res Folder

This folder Contains non-code resources, such as XML layouts, UI strings, and bitmap images, divided into corresponding sub-directories (layout, values, and drawable).

Layout: It has xml files specifying the layout of each activity's UI.

Values: I stored my styled text this sub-directory.

Drawable: I stored all the images and icons used in the project in this sub-directory.

3.2.4 Assets Folder

I stored the text files of pre-stories, stories and post-stories in this sub-directory. Each story, pre-story and post-story text file has the name ID.txt, ID.pre and ID.post where ID is the stop's name. Since the Tour Guide app has content only for one tour I did not need to use SQLite in my app and I was able to handle with files. However, for future work when multiple new tour added to the app it is beneficial to use the SQLite.

4 Main Activity

Purpose:

The main activity is the first activity that will be launched when the app starts. This activity will run in background and does not have a user interface. It sets all the necessary variables, adds the geofences and launches the other three activities (LandingPage, Story and Map) depending on the current values of its state variables. Figure 4 shows the MainActivity class.

Code Base:

The code base of this activity is Geofencing example project, one of the official examples of Android-Play-Location.[7] It uses the Geofencing API to create and remove geofences. After specifying geofences of interest (locations and names), it monitors geofence transitions and creates notifications when the device/user enters or exits a geofence. The code base has five classes, three of which are tailored to Tour Guide's need and are explained in the next three subsections.

4.1 Functionality of the Main Activity

Geofencing Variables (lines 8 - 21):

The geofencing has some variables which I summarize the most important ones.

[7] Downloaded from https://github.com/googlesamples/android-play-location/tree/master/Geofencing

mGeofencingClient is used to get access to the Geofencing API.

mGeofenceList is the list of all geofences of the Tour Guide app.

mAddGeofencesButton and *mRemoveGeofencesButton* are two buttons in the sample geofencing code that when pressed, the geofences will be add/removed to/from the list. In Tour Guide app, the MainActivity does not have a user interface, however, these buttons are pressed by the code when the geofences should be add or removed.

Tour Guide Variables (lines 23 - 40)

uniqueMainActivity is the only instance of MainActivity at any given time. We keep track of that only instance which is used in other activities such as Map.

AssetManager (am) is used to get access to assets such as files. It is used to read the required files (e.g, story files).

History Variables (lines 27 - 34)

lastEnteredGeofence and *lastEexitedGeofence* are the last geofences (if any) that the user has entered or exited.

activeStoryStopName is the stop's name (if any) that its story should be played, i.e., the last visited stop or the stop that the user is interested in.

nextStopTovisit is the next stop (if any) that the user is supposed to visit and the direction is shown to.

requestedStop is the stop (if any) that the user has asked for its direction.

seenStops stores the name of all the stops that the user has visited so far.

State Variables (lines 36 - 40):

showLandingPage is true only when the app should starts the landing page. It is true when the app start and will be update later on.

shouldCoverByMap is true when the app should show the map and false, otherwise.

goToClosetStop is set to true when the user asks for going to the first stop ("start from the nearest stop").

onlyShowMarkers is set when the user asks to show the map without directions ("listen to stories without going on tour").

onCreate function (lines 42 - 77):

This function first removes old geofences (if any). Then, it checks if flags for exit are set by the user and exits the app in that case. It sets uniqueMainActivity, the asset manager (am) and the (hidden) buttons for adding and removing geofences. It also creates an empty list for the geofences and populates them with the geofences.

onStart function (lines 79 - 138):

Each time the main activity (re-)starts, it first makes sure the user has given location permission. Then, it checks the state variables and does one of the following:

- Launches the *LandingPage* activity (lines 90 - 100)
- Launches the *Map* activity to show the map without any direction information (lines 102 - 111)
- Launches the *Map* activity with all the necessary direction information (lines 112 - 124). The geofences also get added by clicking the addGeofenceButton (line 120)
- Launches the *Story* activity (lines 125 - 137)

In all cases, if there is an activity of that specific type running, that activity will be closed. Note that *Intent* is used to create a new activity. The state variables also get updated in a way that the next time the onStart is called (e.g., by minimizing and maximizing the app), the app will work properly.

reset function (lines 140 - 156):

This function resets all the variables of the app to their default value as if the app has started again.

addGeofencesButtonHandler function (lines 158 - 169):

This function gets called when the *mAddGeofencesButton* is pressed. It checks required permissions and adds the geofences by calling *addGeofences* function (lines 175 - 180).

removeGeofencesButtonHandler function (lines 186 - 197):

This function gets called when the *mRemoveGeofencesButton* is pressed. It checks required permissions and adds the geofences by calling *removeGeofences* function (lines 200 - 212).

populateGeofenceList function (lines 214 – 249):

This function creates all the geofences based on their information that is stored in *Constants. Ed_LANDMARKS* and adds them to the *mGeofenceList*.

```
1.  public class MainActivity extends AppCompatActivity implements OnCompleteListener<Voi
    d> {
2.
3.      .
4.      .
5.      .
6.
7.
8.      /**
9.       * Provides access to the Geofencing API.
10.      */
11.     private GeofencingClient mGeofencingClient;
12.
13.     /**
14.      * The list of geofences used in this sample.
15.      */
16.     private ArrayList<Geofence> mGeofenceList;
17.
18.     // Buttons for kicking off the process of adding or removing geofences.
19.     //In Tour Guide app, we only press the buttons in the code and do not show them t
    o the user!
```

```
20.     private Button mAddGeofencesButton;
21.     private Button mRemoveGeofencesButton;
22.
23.     //The only instance of MainActivity
24.     public static MainActivity uniqueMainActivity;
25.     public static AssetManager am;
26.
27.     //history of previous geofence(s)
28.     public static Geofence lastEnteredGeofence = null;
29.     public static Geofence lastExitedGeofence = null;
30.     public static String activeStoryStopName;
31.     public static String nextStopTovisit;
32.     public static String requestedStop = null;//What user asked explicitly. Has prior
    ity over other stops.
33.     public static HashSet<String> seenStops = new HashSet<String>();
34.     public static boolean justEnteredGeofence;
35.
36.     //state variables
37.     public static boolean showLandingPage = true; // should show the landing page?
38.     public static boolean shouldCoverByMap = true;// whether map page should be shown

39.     public static boolean goToClosetStop = false;//false: first stop, true: closest s
    top
40.     public static boolean onlyShowMarkers = false; //true: geofences & direction stop
    s working
41.
42.     @Override
43.     public void onCreate(Bundle savedInstanceState) {
44.         super.onCreate(savedInstanceState);
45.
46.         //remove old geofences (if any). We'll add them later on.
47.         if (MainActivity.uniqueMainActivity!=null){
48.             MainActivity.uniqueMainActivity.mRemoveGeofencesButton.performClick();
49.         }
50.
51.         //handles the case of exiting the app
52.         if (getIntent().getBooleanExtra("EXIT", false)) {
53.             finish();
54.             System.exit(0);
55.         }
56.
57.         //sets the only instance of MainActivity
58.         uniqueMainActivity = this;
59.
60.         //sets the asset manager, which gets used in Util and Constants
61.         am = getAssets();
62.
63.         // Get the UI widgets.
64.         //In Tour Guide project, we only press the buttons and won't show them to the
     user
65.         mAddGeofencesButton = (Button) findViewById(R.id.add_geofences_button);
66.         mRemoveGeofencesButton = (Button) findViewById(R.id.remove_geofences_button);

67.
68.         // Empty list for storing geofences.
69.         mGeofenceList = new ArrayList<>();
70.
71.         // Get the geofences used. Geofence data is hard coded in this sample.
72.         populateGeofenceList();
73.         mGeofencingClient = LocationServices.getGeofencingClient(this);
74.         .
```

```
        .
        .
    }

    @Override
    public void onStart() {
        super.onStart();

        if (!checkPermissions()) {
            requestPermissions();
        }
        else {
            performPendingGeofenceTask();
        }

        if (showLandingPage){
            showLandingPage = false;

            if (LandingPage.uniqueLandingPage !=null){
                LandingPage.uniqueLandingPage.finish();
            }

            Intent intent = new Intent();
            intent.setClass(this, LandingPage.class);
            startActivity(intent);
               }

               //This is for the case of just showing the map, without any direction
s
               else if (shouldCoverByMap && onlyShowMarkers){
                   if (MapsActivityCurrentPlace.uniqueMapsActivityCurrentPlace!=null
){
                       MapsActivityCurrentPlace.uniqueMapsActivityCurrentPlace.finis
h();
                   }

                   Intent intent = new Intent();
                   intent.setClass(this,MapsActivityCurrentPlace.class);
                   startActivity(intent);
               }
               //let's start the Map with all the bits!
               else if (shouldCoverByMap && !onlyShowMarkers){
                   MainActivity.nextStopTovisit = MapsActivityCurrentPlace.getNextSt
op();

                   if (MapsActivityCurrentPlace.uniqueMapsActivityCurrentPlace!=null
){
                       MapsActivityCurrentPlace.uniqueMapsActivityCurrentPlace.finis
h();
                   }

                   mAddGeofencesButton.performClick();
                   Intent intent = new Intent();
                   intent.setClass(this,MapsActivityCurrentPlace.class);
                   startActivity(intent);
               }
               //Lets' start the Story activity otherwise!
               else if (lastEnteredGeofence!=null){
                   mAddGeofencesButton.performClick();
                   activeStoryStopName = lastEnteredGeofence.getRequestId();
                   shouldCoverByMap = true;
```

```
130.                     if (Story.uniqueStory!=null){
131.                         MyTTS.stop();
132.                         Story.uniqueStory.finish();
133.                     }
134.                     justEnteredGeofence = true;
135.                     showStory(this);
136.
137.                 }
138.             }
139.
140.             //reset everything
141.             static void reset(){
142.                 if (MainActivity.uniqueMainActivity!=null){
143.                     MainActivity.uniqueMainActivity.mRemoveGeofencesButton.performCli
    ck();
144.                 }
145.                 lastEnteredGeofence = null;
146.                 lastExitedGeofence = null;
147.                 activeStoryStopName = null;
148.                 nextStopTovisit = MapsActivityCurrentPlace.getNextStop();;
149.                 requestedStop = null;
150.                 shouldCoverByMap = true;
151.                 goToClosetStop = false;
152.                 onlyShowMarkers = false;
153.                 seenStops = new HashSet<String>();
154.                 justEnteredGeofence = false;
155.                 showLandingPage = true;
156.             }
157.
158.             /**
159.              * Adds geofences, which sets alerts to be notified when the device enter
    s or exits one of the
160.              * specified geofences. Handles the success or failure results returned b
    y addGeofences().
161.              */
162.             public void addGeofencesButtonHandler(View view) {
163.                 if (!checkPermissions()) {
164.                     mPendingGeofenceTask = PendingGeofenceTask.ADD;
165.                     requestPermissions();
166.                     return;
167.                 }
168.                 addGeofences();
169.             }
170.
171.             /**
172.              * Adds geofences. This method should be called after the user has grante
    d the location
173.              * permission.
174.              */
175.             @SuppressWarnings("MissingPermission")
176.             private void addGeofences() {
177.                 if (!checkPermissions()) {
178.                     showSnackbar(getString(R.string.insufficient_permissions));
179.                     return;
180.                 }
181.
182.                 mGeofencingClient.addGeofences(getGeofencingRequest(), getGeofencePen
    dingIntent())
183.                         .addOnCompleteListener(this);
184.             }
185.
```

```
186.            /**
187.             * Removes geofences, which stops further notifications when the device e
    nters or exits
188.             * previously registered geofences.
189.             */
190.            public void removeGeofencesButtonHandler(View view) {
191.                if (!checkPermissions()) {
192.                    mPendingGeofenceTask = PendingGeofenceTask.REMOVE;
193.                    requestPermissions();
194.                    return;
195.                }
196.                removeGeofences();
197.            }
198.
199.
200.            /**
201.             * Removes geofences. This method should be called after the user has gra
    nted the location
202.             * permission.
203.             */
204.            @SuppressWarnings("MissingPermission")
205.            private void removeGeofences() {
206.                if (!checkPermissions()) {
207.                    showSnackbar(getString(R.string.insufficient_permissions));
208.                    return;
209.                }
210.
211.                mGeofencingClient.removeGeofences(getGeofencePendingIntent()).addOnCo
    mpleteListener(this);
212.            }
213.
214.            /**
215.             * This sample hard codes geofence data. A real app might dynamically cre
    ate geofences based on
216.             * the user's location.
217.             */
218.            private void populateGeofenceList() {
219.                for (Map.Entry<String, LatLng> entry : Constants.Ed_LANDMARKS.entrySe
    t()) {
220.
221.                    mGeofenceList.add(new Geofence.Builder()
222.                            // Set the request ID of the geofence. This is a string t
    o identify this
223.                            // geofence.
224.                            .setRequestId(entry.getKey())
225.
226.                            // Set the circular region of this geofence.
227.                            .setCircularRegion(
228.                                    entry.getValue().latitude,
229.                                    entry.getValue().longitude,
230.                                    Constants.GEOFENCE_RADIUS_IN_METERS
231.                            )
232.
233.                            // Set the expiration duration of the geofence. This geof
    ence gets automatically
234.                            // removed after this period of time.
235.                            .setExpirationDuration(Constants.GEOFENCE_EXPIRATION_IN_M
    ILLISECONDS)
236.
237.                            // Set the transition types of interest. Alerts are only
    generated for these
```

```
238.                            // transition. We track entry and exit transitions in thi
    s sample.
239.                            .setTransitionTypes(Geofence.GEOFENCE_TRANSITION_ENTER |
240.                                    Geofence.GEOFENCE_TRANSITION_EXIT)
241.
242.                            // Create the geofence.
243.                            .build());
244.                }
245.            }
246.
247.            .
248.            .
249.            .
250.        }
```

Figure 4. MainActivity class

4.2 Constants class

Purpose:

The constants class is used to set the hyper-parameters of geofencing as well as the information of geofences (their names, locations and stories). It also loads numbered markers' images (red and green).Figure 5 shows the Constants class.

Parameters (lines 3 – 20):

GEOFENCE_EXPIRATION_IN_HOURS (*GEOFENCE_EXPIRATION_IN_MILLISECONDS)* is the number of hours (milliseconds) that the geofences will expire after that. The duration that I used is 12 hours, meaning that the Tour Guide app will no longer track the geofences after 12 hours.

GEOFENCE_RADIUS_IN_METERS is the radius (in meters) of geofences which is set to *35*.

MINWRONGDIST is used to specify when a user is considered lost: In our case, a user is considered as lost if s(he) deviates from the suggested route more than *70* meters.

NUMWRONGDIRLIST is the number of times the app waits before giving the "wrong direction" alert. This parameter is set to *3*.

Data structures for storing the tour's information (lines 22 – 34):

ED_LANDMARKS stores a map from the names of the stops (ids of geofences) to their LatLong (latitude and longitude)

id2story, *id2preStory* and *id2postStory* contain a map from each stop name to their stories, pre-stories, and post-stories.

seenStop2Sign and *unSeenStop2Sign* are maps from markers numbers (0, 1, 2, etc) to saved images of them with colors green and red, respectively.

Static block to fill the data structures (lines 36 - 87):

A static block (specified by *static* keyword before) gets called only once in the app's lifetime. In Tour Guide app, a static class is used to load all the necessary information about the ghost tour. It first sets *ED_LANDMARKS* (lines 39 - 47), then sets *unSeenStop2Sign* and *seenStop2Sign* (lines 49 - 60). Finally, it loads all the stories, pre-stories and post-stories (lines 65 – 86).

```
1.  final class Constants {
2.
3.      /**
4.       * Used to set an expiration time for a geofence. After this amount of time Locat
    ion Services
5.       * stops tracking the geofence.
6.       */
7.      private static final long GEOFENCE_EXPIRATION_IN_HOURS = 12;
8.
9.      /**
10.      * For this sample, geofences expire after twelve hours.
11.      */
12.     static final long GEOFENCE_EXPIRATION_IN_MILLISECONDS =
13.             GEOFENCE_EXPIRATION_IN_HOURS * 60 * 60 * 1000;
14.
15.     //Geofences' radius
16.     static final float GEOFENCE_RADIUS_IN_METERS = 35;
17.
18.     //Parameters for coping with getting lost
19.     public static double MINWRONGDIST = 70;
20.     public static int NUMWRONGDIRLIST = 3;
21.
22.     /**
23.      * data structures for storing information about story stops in the Edinburgh.
24.      */
25.     static final LinkedHashMap<String, LatLng> Ed_LANDMARKS = new LinkedHashMap<>();

26.     public static HashMap<String,String> id2Story = new HashMap<>();
27.     public static HashMap<String,String> id2PostStory = new HashMap<>();
28.     public static HashMap<String,String> id2PreStory = new HashMap<>();
29.     public static ArrayList<String> ids = new ArrayList<>();
30.     public static HashMap<String,Integer> id2Idx = new HashMap<>();
31.
32.     //a hashMap from numbers (0-
    26) to numbers (R.drawable.a...) to handle red and green colors
33.     static final HashMap<Integer, Integer> unSeenStop2Sign = new HashMap<>();
34.     static final HashMap<Integer, Integer> seenStop2Sign = new HashMap<>();
35.
36.     //static block called when the project gets loaded
37.
38.     static {
39.         Ed_LANDMARKS.put("Edinburgh Castle",new LatLng(55.948606, -3.198143));
40.         Ed_LANDMARKS.put("Castlehill",new LatLng(55.948786, -3.196796));
41.         Ed_LANDMARKS.put("Lawnmarket and Victoria Street",new LatLng(55.948792, -
    3.193799));
42.         Ed_LANDMARKS.put("Mary King's Close",new LatLng(55.949873, -3.190470));
43.         Ed_LANDMARKS.put("Mid-High Street",new LatLng(55.950202, -3.187189));
44.         Ed_LANDMARKS.put("Lower High Street",new LatLng(55.950332, -3.186279));
45.         Ed_LANDMARKS.put("Canongate I",new LatLng(55.950702, -3.183809));
46.         Ed_LANDMARKS.put("Canongate II",new LatLng(55.951503, -3.179397));
47.         Ed_LANDMARKS.put("Holyrood",new LatLng(55.952736, -3.173873));
48.
49.         //load the unseen stops images
```

```
50.         unSeenStop2Sign.put(0,R.drawable.a);
51.         unSeenStop2Sign.put(1,R.drawable.b);
52.         unSeenStop2Sign.put(2,R.drawable.c);
53.         .
54.         .
55.         .
56.
57.         //load the seen stops images
58.         seenStop2Sign.put(0,R.drawable.ag);
59.         seenStop2Sign.put(1,R.drawable.bg);
60.         seenStop2Sign.put(2,R.drawable.cg);
61.         .
62.         .
63.         .
64.
65.         //the root directory having all the stories
66.         File dir = new File("Ghost_Tour/");
67.
68.         int idx = 0;
69.         //load the stories, pre-stories and post-stories
70.         for (String id: Ed_LANDMARKS.keySet()){
71.             String storyFileName =  id+".story";
72.             String preStoryFileName =  id+".pre";
73.             String postSToryFileName = id+".dir";
74.
75.             String story = Util.readFileFromAsset(storyFileName);
76.             String preStory = Util.readFileFromAsset(preStoryFileName);
77.             String postStory = Util.readFileFromAsset(postSToryFileName);
78.
79.             id2Story.put(id,story);
80.             id2PreStory.put(id,preStory);
81.             id2PostStory.put(id,postStory);
82.             id2Idx.put(id,idx);
83.             ids.add(id);
84.
85.             idx++;
86.         }
87.     }
88. }
```

Figure 5. Constants class

4.3 GeofenceTransitionsJobIntentService class

Purpose:

This class is used to handle geofence transitions (enter and exit). While the class has a number of variables and methods, we have changed only one of its methods which is described below. Figure 6 shows the code for this class.

onHandleWork (lines 7 - 96):

This function handles an enter/exit event to/from a geofence. It first handles possible errors (lines 14 - 20). Then if the transition is enter or exit (not dwelling), it gets the list of transitions (almost always only one

transition) (line 30). It forms the notification message (line 33). Then, one of the two following cases happens:

- Entering a geofence (lines 37 - 63): It gets the geofence's information (line 38) and checks whether it is a repeated geofence, in which case the function will be terminated (lines 40 - 44). It then updates the state variables of the main activity (lines 46 - 58) including the visited stops (seenStops) and the name of the last entered geofence. Finally, it plays the entrance message by TTS (line 61 - 62).
- Exiting a geofence (lines 64 - 86): Similar to the entrance case, it first checks whether it is a repeated geofence to terminates the function (lines 66 - 72). Otherwise, it updates the state variables of the main activity (lines 74 - 75), gets the device's location and updates the direction (line 77 to the next stop). Finally, it plays the exit message by TTS (lines 79 – 85).

Finally, it gives entrance/exit notification (line 89).

```
1.  public class GeofenceTransitionsJobIntentService extends JobIntentService {
2.
3.      .
4.      .
5.      .
6.
7.      /**
8.       * Handles incoming intents.
9.       * @param intent sent by Location Services. This Intent is provided to Location
10.      *               Services (inside a PendingIntent) when addGeofences() is called.
11.      */
12.     @Override
13.     protected void onHandleWork(Intent intent) {
14.         GeofencingEvent geofencingEvent = GeofencingEvent.fromIntent(intent);
15.         if (geofencingEvent.hasError()) {
16.             String errorMessage = GeofenceErrorMessages.getErrorString(this,
17.                     geofencingEvent.getErrorCode());
18.             Log.e(TAG, errorMessage);
19.             return;
20.         }
21.
22.         // Get the transition type.
23.         int geofenceTransition = geofencingEvent.getGeofenceTransition();
24.
25.         // Test that the reported transition was of interest.
26.         if (geofenceTransition == Geofence.GEOFENCE_TRANSITION_ENTER ||
27.                 geofenceTransition == Geofence.GEOFENCE_TRANSITION_EXIT) {
28.
29.             // Get the geofences that were triggered. A single event can trigger mult
    iple geofences.
30.             List<Geofence> triggeringGeofences = geofencingEvent.getTriggeringGeofenc
    es();
31.
32.             // Get the transition details as a String.
33.             String geofenceTransitionDetails = getGeofenceTransitionDetails(geofenceT
    ransition,
34.                     triggeringGeofences);
35.
36.
37.             if (geofenceTransition == Geofence.GEOFENCE_TRANSITION_ENTER){
```

```
                Geofence thisGeofence = triggeringGeofences.get(triggeringGeofences.s
ize()-1);

                //repeated geofence, just ignore
                if (MainActivity.lastEnteredGeofence != null && MainActivity.lastEnte
redGeofence.
                        getRequestId().equals(thisGeofence.getRequestId())){
                    return ;
                }

                //setting the state variables
                MainActivity.lastEnteredGeofence = thisGeofence;
                MainActivity.activeStoryStopName = thisGeofence.getRequestId();
                MainActivity.shouldCoverByMap=false;
                for (Geofence geofence : triggeringGeofences) {
                    MainActivity.seenStops.add(geofence.getRequestId());
                }

                //record when the user gets to the requested stop
                if (thisGeofence.getRequestId().equals(MainActivity.requestedStop)){

                    //Yayy, finally got to the requested stop.
                    MainActivity.requestedStop = null;
                }

                //playing the sound for entering a geofence
                MyTTS.play("You have arrived at " +  MainActivity.activeStoryStopName
 +
                        "Please press the notification message to listen to the story
.",false);
            }
            else if (geofenceTransition == Geofence.GEOFENCE_TRANSITION_EXIT){

                Geofence thisGeofence = triggeringGeofences.get(triggeringGeofences.s
ize()-1);

                //repeated geofence, just ignore it
                if (MainActivity.lastExitedGeofence != null && MainActivity.lastExite
dGeofence.
                        getRequestId().equals(thisGeofence.getRequestId())){
                    return ;
                }

                MainActivity.lastExitedGeofence = thisGeofence;
                MainActivity.shouldCoverByMap=true;

                MapsActivityCurrentPlace.uniqueMapsActivityCurrentPlace.getDeviceLoca
tion(true);

                String nextStopTovisit = MapsActivityCurrentPlace.getNextStop();
                if (MainActivity.requestedStop==null || MainActivity.requestedStop.
                        equals(nextStopTovisit)){
                    //let the user know about exiting
                    MyTTS.play("you have exited the" +  thisGeofence.getRequestId() +

                            "If you need an updated direction, please see the map.",f
alse);
                }
            }

            // Send notification and log the transition details.
```

```
89.             sendNotification(geofenceTransitionDetails);
90.             Log.i(TAG, geofenceTransitionDetails);
91.         } else {
92.             // Log the error.
93.             Log.e(TAG, getString(R.string.geofence_transition_invalid_type, geofenceT
    ransition));
94.         }
95.     }
96. }
```

Figure 6. GeofenceTransitionsJobIntentService class

5 LandingPage Activity

Purpose:

This activity forms the landing page of the app, i.e., the page that is shown to the user when the app starts. It contains three main tabs: Introduction, Tour, and History (Research Paper, Section 4). It assigns the necessary functionalities to the tabs so that the user can interact with them.

Code Base:

The code base of this activity is a new project of type Tabbed Activity/Action Bar Tabs (with ViewPager).[8] I first created a project with the desired tabs, then merged it with the main project.

5.1 User Interface (landingpage_activity.xml)

The user interface of the landing page is determined by landingpage_activity.xml (Figure 7). Figure 8 shows the main layout graphically. The landing page has three tabs that are defined here as three placeholders. However, the exact functionality and UI of the tabs are defined using three fragments (Sections 5.3, 5.4, 5.5)

The main layout is a ConstraintLayout that is usually used when the content of a page has a pre-specified shape. It has the following components:

High level properties (lines 3 - 14)

The high-level properties of the layout such as its size and position are specified.

AppBarLayout (lines 17 - 52)

The AppBarLayout (appbar) defines the layout of the horizontal bar that contains the tabs.

TabLayout (lines 25 - 50**)**

The TabLayout (tabs) defines the three used tabs and assigns ids to them.

ImageView (lines 54 - 68)

[8] More information in the following short tutorial: https://www.youtube.com/watch?v=00LLd7qr9sA

The ImageView (stopImage) defines the position and the id of the image that is placed at the top of the landing page.

ViewPager (lines 70 - 83)

The ViewPager (container) allocates some space that is scrollable (if necessary) and will be later on filled out with the associated Fragment of each tab.

```
1.  <?xml version="1.0" encoding="utf-8"?>
2.
3.  <android.support.constraint.ConstraintLayout xmlns:android="http://schemas.android.co
    m/apk/res/android"
4.      xmlns:app="http://schemas.android.com/apk/res-auto"
5.      xmlns:tools="http://schemas.android.com/tools"
6.      android:id="@+id/main_content"
7.      android:layout_width="match_parent"
8.      android:layout_height="match_parent"
9.      android:fitsSystemWindows="true"
10.     app:layout_editor_absoluteX="0dp"
11.     app:layout_editor_absoluteY="80dp"
12.     tools:context=".LandingPage"
13.     tools:layout_editor_absoluteX="0dp"
14.     tools:layout_editor_absoluteY="80dp">
15.
16.
17.     <android.support.design.widget.AppBarLayout
18.         android:id="@+id/appbar"
19.         android:layout_width="match_parent"
20.         android:layout_height="wrap_content"
21.         android:theme="@style/AppTheme.AppBarOverlay"
22.         app:layout_constraintEnd_toEndOf="parent"
23.         app:layout_constraintTop_toBottomOf="@+id/stopImage">
24.
25.         <android.support.design.widget.TabLayout
26.             android:id="@+id/tabs"
27.             android:layout_width="match_parent"
28.             android:layout_height="match_parent"
29.             android:background="@color/colorPrimary"
30.             app:tabIndicatorColor="@color/colorAccent">
31.
32.             <android.support.design.widget.TabItem
33.                 android:id="@+id/tabItem"
34.                 android:layout_width="wrap_content"
35.                 android:layout_height="wrap_content"
36.                 android:text="@string/tab_text_1" />
37.
38.             <android.support.design.widget.TabItem
39.                 android:id="@+id/tabItem2"
40.                 android:layout_width="wrap_content"
41.                 android:layout_height="wrap_content"
42.                 android:text="@string/tab_text_2" />
43.
44.             <android.support.design.widget.TabItem
45.                 android:id="@+id/tabItem3"
46.                 android:layout_width="wrap_content"
47.                 android:layout_height="wrap_content"
48.                 android:text="@string/tab_text_3" />
49.
```

```
50.         </android.support.design.widget.TabLayout>
51.
52.     </android.support.design.widget.AppBarLayout>
53.
54.     <ImageView
55.         android:id="@+id/stopImage"
56.         android:layout_width="380dp"
57.         android:layout_height="261dp"
58.         android:layout_marginEnd="8dp"
59.         android:layout_marginRight="8dp"
60.         android:adjustViewBounds="false"
61.         android:contentDescription="@string/app_name"
62.         android:cropToPadding="false"
63.         app:layout_constraintBottom_toTopOf="@+id/appbar"
64.         app:layout_constraintEnd_toEndOf="parent"
65.         app:layout_constraintHorizontal_bias="0.552"
66.         app:layout_constraintStart_toStartOf="parent"
67.         app:layout_constraintTop_toTopOf="parent"
68.         app:srcCompat="@drawable/royalmile" />
69.
70.     <android.support.v4.view.ViewPager
71.         android:id="@+id/container"
72.         android:layout_width="359dp"
73.         android:layout_height="244dp"
74.         android:layout_marginBottom="8dp"
75.         android:layout_marginEnd="12dp"
76.         android:layout_marginStart="12dp"
77.         android:layout_marginTop="4dp"
78.         app:layout_behavior="@string/appbar_scrolling_view_behavior"
79.         app:layout_constraintBottom_toBottomOf="parent"
80.         app:layout_constraintEnd_toEndOf="parent"
81.         app:layout_constraintStart_toStartOf="parent"
82.         app:layout_constraintTop_toBottomOf="@+id/appbar"
83.         app:layout_constraintVertical_bias="0.100000024" />
84.
85. </android.support.constraint.ConstraintLayout>
```

Figure 7. LandingPage activity layout XML file

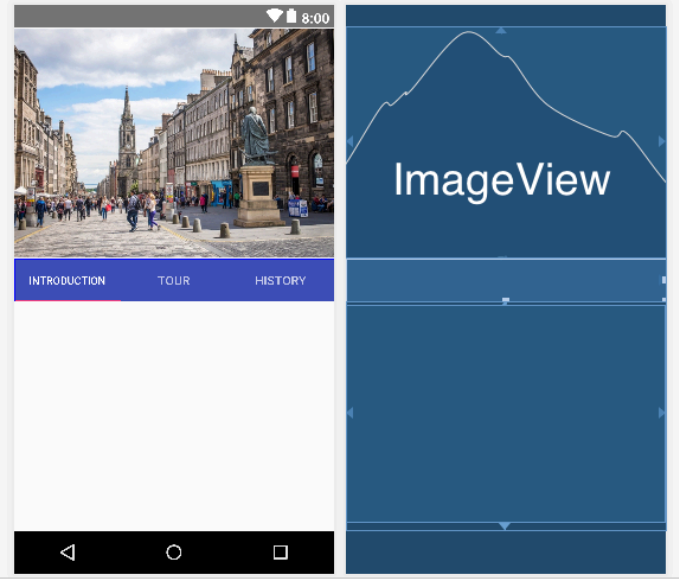


Figure 8. LandingPage Activity Layout

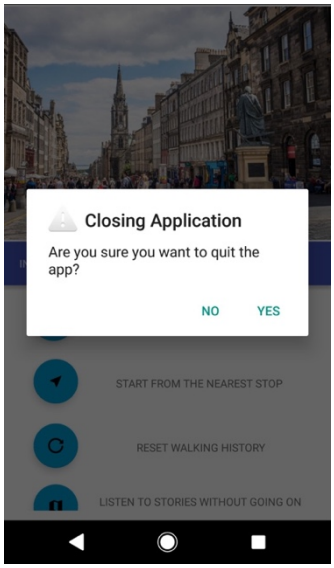


Figure 9. Closing the app

5.2 Functionality

Figure 10 shows the code for this activity.

Variables (lines 7 - 16):

uniqueLandingPage is the only instance of LandingPage at any given time. We keep track of that only instance which gets used in the MainActivity.

mSectionsPagerAdapter is an adapter that has the necessary information about the three tabs. It is of type *SectionsPagerAdapter* (lines 47 - 95), which is an inner class containing the information about the tabs, i.e., the number of tabs (3), their names ("INTRODUCTION", "TOUR" and "HISTROY") and their relative order (0: Tab1, 1: Tab2 and 2: Tab3).

mViewPager hosts the content of the tabs, i.e., what each tab shows.

onCreate function (lines 18 - 41):

The function first instantiates the above variables. It loads the layout of the tabs (tabLayout) and adds the necessary functionality to it for cases when the user changes between the tabs. It makes sure that the TTS stops playing contents (if it is already playing) when the user presses another tab. This functionality is gained by use of the implemented class *MyTabLayoutViewPagerOnTabSelectedListener*.

SectionsPagerAdapter (lines 47 - 95):

This is an inner class which is explained while discussing *mSectionsPagerAdapter* variable.

onBackPressed (lines 97 - 118):

This function is called when the user presses back while in the landing page. Since the LandingPage activity is the first page that the user sees when s(he) opens the app, pressing back from this page means that s(he) might want to quit the app. In that case, the app shows an alert message that asks “Are you sure you want to quit the app?” (Figure 9). The user can press “No” and stay at same tab in the landing page or “Yes” and quit the app.

class MyTabLayoutViewPagerOnTabSelectedListener (lines 121 - 141):

This class is used to perform one extra action when the user changes the tab: It stops the TTS (if it is already plaing). This is done by extending the class TabLayout.ViewPagerOnTabSelectedListener, which is used by default to handle changes between the tabs, and re-implementing its onTabSelected function.

When I wanted to build a UI I either wrote the layout xml by hand or dragged the UI elements into the blueprint visual editor.

```
1.  public class LandingPage extends AppCompatActivity {
2.
3.      .
4.      .
5.      .
6.
7.      //The only instance of LandingPage, to be used in MainActivity
8.      public static LandingPage uniqueLandingPage;
9.
10.     //the adapter that will return a fragment for each of the three primary sections
    of the activity.
11.     private SectionsPagerAdapter mSectionsPagerAdapter;
12.
13.     /*
14.      * The {@link ViewPager} that will host the section contents.
15.      */
16.     private ViewPager mViewPager;
17.
18.     @Override
19.     protected void onCreate(Bundle savedInstanceState) {
20.
21.         .
22.         .
23.         .
24.
25.         uniqueLandingPage = this;
26.
27.         // Create the adapter
28.         mSectionsPagerAdapter = new SectionsPagerAdapter(getSupportFragmentManager())
    ;
29.
30.         // Set up the ViewPager with the sections adapter.
31.         mViewPager = (ViewPager) findViewById(R.id.container);
32.         mViewPager.setAdapter(mSectionsPagerAdapter);
33.
34.         // Get the tab layout and link it to the mViewPager
35.         TabLayout tabLayout = (TabLayout) findViewById(R.id.tabs);
36.         mViewPager.addOnPageChangeListener(new TabLayout.TabLayoutOnPageChangeListene
    r(tabLayout));
37.
38.         // Set up the lister for tab selection
```

```
39.         TabLayout.ViewPagerOnTabSelectedListener tb = new MyTabLayoutViewPagerOnTabSe
    lectedListener(mViewPager);
40.         tabLayout.addOnTabSelectedListener(tb);
41.     }
42.
43.     .
44.     .
45.     .
46.
47.     /**
48.      * A {@link FragmentPagerAdapter} that returns a fragment corresponding to
49.      * one of the sections/tabs/pages.
50.      */
51.     public class SectionsPagerAdapter extends FragmentPagerAdapter {
52.
53.         public SectionsPagerAdapter(FragmentManager fm) {
54.             super(fm);
55.         }
56.
57.         @Override
58.         public Fragment getItem(int position) {
59.
60.             switch (position) {
61.                 case 0:
62.                     Tab1 tab1 = new tab1();
63.                     return tab1;
64.                 case 2:
65.                     Tab2 tab2 = new tab2();
66.                     return tab2;
67.
68.                 case 1:
69.                     Tab3 tab3 = new tab3();
70.                     return tab3;
71.
72.                 default:
73.                     return null;
74.             }
75.         }
76.
77.         @Override
78.         public int getCount() {
79.             // Show 3 total pages.
80.             return 3;
81.         }
82.
83.         @Override
84.         public CharSequence getPageTitle(int position) {
85.             switch (position) {
86.                 case 0:
87.                     return "INTRODUCTION";
88.                 case 1:
89.                     return "TOUR";
90.                 case 2:
91.                     return "HISTORY";
92.             }
93.             return null;
94.         }
95.     }
96.
97.     //Added to handle quiting the app
98.     @Override
```

```
99.     public void onBackPressed() {
100.                new AlertDialog.Builder(this)
101.                        .setIcon(android.R.drawable.ic_dialog_alert)
102.                        .setTitle("Closing Application")
103.                        .setMessage("Are you sure you want to quit the app?")
104.                        .setPositiveButton("Yes", new DialogInterface.OnClickListener
    ()
105.                        {
106.                            @Override
107.                            public void onClick(DialogInterface dialog, int which) {

108.                                //finishing everything
109.                                MyTTS.stop();
110.                                Intent intent = new Intent(getApplicationContext(), M
    ainActivity.class);
111.                                intent.setFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
112.                                intent.putExtra("EXIT", true);
113.                                startActivity(intent);
114.                            }
115.                        })
116.                        .setNegativeButton("No", null)
117.                        .show();
118.            }
119. }
120.
121.        /* A simple extention of TabLayout.ViewPagerOnTabSelectedListener, because I
    wanted to override
122.        its onTabSelected to stop the MyTTS
123.         */
124.        class MyTabLayoutViewPagerOnTabSelectedListener extends TabLayout.ViewPagerOn
    TabSelectedListener{
125.
126.            public MyTabLayoutViewPagerOnTabSelectedListener(ViewPager mViewPager){
127.                super(mViewPager);
128.            }
129.
130.            @Override
131.            public void onTabSelected(TabLayout.Tab tab) {
132.                System.out.println("switching tabs");
133.                MyTTS.stop();
134.                super.onTabSelected(tab);
135.            }
136.
137.            .
138.            .
139.            .
140.
141. }
```

Figure 10. LandingPage Activity class

5.3 First Tab: Introduction Fragment

Each tab of the landing page is defined by a fragment, which is used to control the behavior of a portion of UI. Multiple fragments can be combined in an activity to build a multi-pane UI (AndroidDevelopers, 2018e).

5.3.1 User Interface

The user interface of the introduction fragment is determined by tab1.xml (Figure 11). Figure 12 shows the main layout graphically.

The main layout is a ConstraintLayout and has the following components:

High level properties (lines 3 - 8):

The high-level properties of the layout such as its size which is matched with its parent (the placeholder that contains the fragment).

ScrollView (lines 10 - 58):

It makes the content of the fragment scrollable. It consists of a *LinearLayout* that has the two buttons (play and stop) and the text.

ConstraintLayout (lines 22 - 46):

This is used to define the exact positions of the play and stop buttons

FloatingActionButton (Play) (lines 26 - 34):

It defines the play button, its name, position, shape and the function

FloatingActionButton (Stop) (lines 36 - 44):

It defines the stop button, its name, position, shape and the function

TextView (overview_text) (lines 48 - 54):
Defines the styled text that will be shown for introduction.

```
1.  <?xml version="1.0" encoding="utf-8"?>
2.  <android.support.constraint.ConstraintLayout xmlns:android="http://schemas.android.co
    m/apk/res/android"
3.      xmlns:app="http://schemas.android.com/apk/res-auto"
4.      xmlns:tools="http://schemas.android.com/tools"
5.      android:id="@+id/constraintLayout"
6.      android:layout_width="match_parent"
7.      android:layout_height="match_parent"
8.      tools:context=".Introduction$PlaceholderFragment">
9.
10.     <ScrollView
11.         android:layout_width="367dp"
12.         .
13.         .
14.         .
15.         >
16.
17.         <LinearLayout
18.             android:layout_width="match_parent"
19.             android:layout_height="wrap_content"
20.             android:orientation="vertical">
```

```
21.
22.             <android.support.constraint.ConstraintLayout
23.                 android:layout_width="match_parent"
24.                 android:layout_height="72dp">
25.
26.                 <android.support.design.widget.FloatingActionButton
27.                     android:id="@+id/play"
28.                     android:layout_width="64dp"
29.                     .
30.                     .
31.                     .
32.                     android:clickable="true"
33.                     android:src="@drawable/play"
34.                     />
35.
36.                 <android.support.design.widget.FloatingActionButton
37.                     android:id="@+id/stop"
38.                     android:layout_width="64dp"
39.                     .
40.                     .
41.                     .
42.                     android:clickable="true"
43.                     android:src="@drawable/stop"
44.                     />
45.
46.             </android.support.constraint.ConstraintLayout>
47.
48.             <TextView
49.                 android:id="@+id/overview_text"
50.                 android:layout_width="match_parent"
51.                 android:text="@string/tab1_styled_text"
52.                 .
53.                 .
54.                 .
55.             />
56.
57.         </LinearLayout>
58.     </ScrollView>
59.
60. </android.support.constraint.ConstraintLayout>
```

Figure 11. Introduction fragment layout XML file

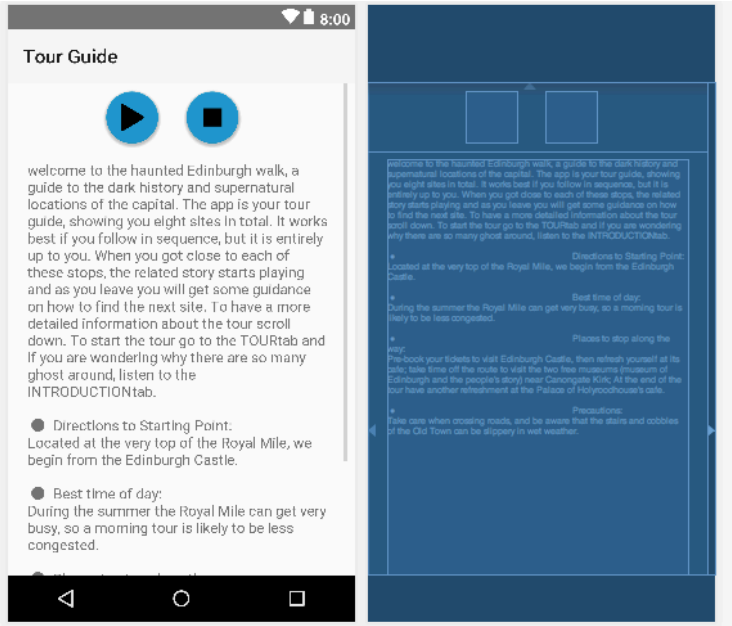


Figure 12. Introduction fragment layout

5.3.2 Functionality

Figure 13 shows the code for this class.

Purpose:

The introduction fragment forms the first tab.

Implemented Interface(s) (line 1):

This class implements the View.OnClickListener interface. Any class that implements this interface must implement an onClick function. As a result, the objects in that class can be used to assign functionality to UI elements such as buttons (see onCreateView and onClick below for more information).

OnCreateView (lines 3 - 16):

This function gets called to perform necessary initialization for the introduction tab. In particular, it determines that when *play* and *stop* button are clicked, the onClick function will be called.

OnStart (lines 18 - 22):

The onStart function gets called whenever the user clicks the introduction tab. It stops the TTS if it is already playing.

onClick (lines 24 - 36):

The onClick function specifies that a) if the play button is clicked, a text will be played by the TTS; b) if the stop button is pressed, the TTS will stop playing.

```
1.  public class Tab1 extends Fragment implements View.OnClickListener {
2.
3.      @Override
4.      public View onCreateView(LayoutInflater inflater, ViewGroup container,
5.                               Bundle savedInstanceState) {
6.          .
7.          .
8.          .
9.
10.         FloatingActionButton play = (FloatingActionButton) rootView.findViewById(R.id
    .play);
11.         FloatingActionButton stop = (FloatingActionButton) rootView.findViewById(R.id
    .stop);
12.         play.setOnClickListener(this);
13.         stop.setOnClickListener(this);
14.         return rootView;
15.
16.     }
17.
18.     @Override
19.     public void onStart() {
20.         super.onStart();
21.         MyTTS.stop();
22.     }
23.
24.     //this is implemented as the listener
25.     @Override
26.     public void onClick(View rootView) {
27.
28.         switch (rootView.getId()) {
29.             case R.id.play:
30.                 MyTTS.play("welcome to the haunted Edinburgh walk, a guide to the dar
    k ....")
31.                 break;
32.             case R.id.stop:
33.                 MyTTS.stop();
34.                 break;
35.         }
36.     }
37. }
```

Figure 13. Introduction fragment

5.3.3 Content

This tab contains information about: Where to begin the tour; Best time of the day for having it, Free museums on the tour's route; and last but not least, safety tips for the user.

Content of the Introduction Tab:

Welcome to the haunted Edinburgh walk, a guide to the dark history and supernatural locations of the capital. The app is your tour guide, showing you nine sites in total. It works best if you follow in sequence, but it is entirely up to you. When you got close to each of these stops, the related story starts playing and as you leave you will get some guidance on how to find the next site.
To have detailed information about the tour scroll down. To start the tour go to the "TOUR" tab, and if you are wondering why there are so many ghost around listen to the "HISTORY" tab.

- **Directions to Starting Point:**
  Located at the very top of the Royal Mile, we begin from the Edinburgh Castle.

- **Best time of day:**
  During the summer the Royal Mile can get very busy, so a morning tour is likely to be less congested.

- **Places to stop along the way:**
  Pre-book your tickets to visit Edinburgh Castle, then refresh yourself at its cafe; take time off the route to visit the two free museums (museum of Edinburgh and the people's story) near Canongate Kirk; At the end of the tour have another refreshment at the Palace of Holyrood house's cafe.

- **Precautions:**
  Take care when crossing roads and be aware that the stairs and cobbles of the Old Town can be slippery in wet weather.

5.4 Second Tab: Tour Fragment

5.4.1 User Interface

The user interface of the Tour fragment is determined by tab2.xml (Figure 14). Figure 15 shows the main layout graphically. This tab has four buttons (e.g., “take me to the first site”) and their descriptions, then a *preview* text and a *preview* image.

The main layout is a ConstraintLayout and has the following components:

High level properties (lines 3 - 8):

The high-level properties of the layout such as its size which is matched with its parent (the placeholder that contains the fragment).

ScrollView (lines 10 - 159):

It makes the content of the fragment scrollable. It consists of a *LinearLayout* that has everything in this layout.

ConstraintLayout (lines 21 – 47, lines 49 – 75, lines 77 – 101, lines 103 - 125):

This is used to define the exact positions of the four buttons and their descriptions. Each button is defined using a *FloatingActionButton* (e.g., lines 27 – 36 for ) and a description text (e.g., "take me to the first site").

ConstraintLayout (lines l27 – 141, lines 143 -156):

These ConstraintLayouts define "preview" text and image, respectively.

```
1.  <?xml version="1.0" encoding="utf-8"?>
2.  <android.support.constraint.ConstraintLayout xmlns:android="http://schemas.android.co
    m/apk/res/android"
3.      xmlns:app="http://schemas.android.com/apk/res-auto"
4.      xmlns:tools="http://schemas.android.com/tools"
5.      android:id="@+id/constraintLayout"
6.      android:layout_width="match_parent"
7.      android:layout_height="match_parent"
8.      tools:context=".Introduction$PlaceholderFragment">
9.
10.     <ScrollView
11.         android:layout_width="367dp"
12.         .
13.         .
14.         .>
15.
16.         <LinearLayout
17.             android:layout_width="match_parent"
18.             android:layout_height="wrap_content"
19.             android:orientation="vertical">
20.
21.             <android.support.constraint.ConstraintLayout
22.                 android:layout_width="338dp"
23.                 .
24.                 .
25.                 .>
26.
27.                 <android.support.design.widget.FloatingActionButton
28.                     android:id="@+id/startingDir"
29.                     android:layout_width="80dp"
30.                     .
31.                     .
32.                     .
33.                     android:clickable="true"
34.                     android:focusable="true"
35.                     android:src="@drawable/navigation"
36.                     />
37.
38.                 <TextView
39.                     android:id="@+id/textView3"
40.                     android:layout_width="241dp"
```

```
41.                     .
42.                     .
43.                     .
44.                     android:text="@string/first_stop"
45.                     />
46.
47.             </android.support.constraint.ConstraintLayout>
48.
49.             <android.support.constraint.ConstraintLayout
50.                 android:layout_width="338dp"
51.                 .
52.                 .
53.                 .>
54.
55.                 <android.support.design.widget.FloatingActionButton
56.                     android:id="@+id/closestDir"
57.                     android:layout_width="80dp"
58.                     .
59.                     .
60.                     .
61.                     android:clickable="true"
62.                     android:focusable="true"
63.                     android:src="@drawable/nearme"
64.                  />
65.
66.                 <TextView
67.                     android:id="@+id/textView4"
68.                     android:layout_width="241dp"
69.                     .
70.                     .
71.                     .
72.                     android:text="@string/closest_stop"
73.                  />
74.
75.             </android.support.constraint.ConstraintLayout>
76.
77.             <android.support.constraint.ConstraintLayout
78.                 android:layout_width="338dp"
79.                 .
80.                 .
81.                 .>
82.
83.                 <android.support.design.widget.FloatingActionButton
84.                     android:id="@+id/resetTour"
85.                     android:layout_width="80dp"
86.                     .
87.                     .
88.                     .
89.                     android:clickable="true"
90.                     android:focusable="true"
91.                     />
92.
93.                 <TextView
94.                     android:id="@+id/textView5"
95.                     android:layout_width="241dp"
96.                     .
97.                     .
98.                     .
99.                     android:text="@string/reset"/>
100.
101.            </android.support.constraint.ConstraintLayout>
```

```
102.
103.            <android.support.constraint.ConstraintLayout
104.                android:layout_width="338dp"
105.                .
106.                .
107.                .>
108.
109.                <android.support.design.widget.FloatingActionButton
110.                    android:id="@+id/mapShow"
111.                    android:clickable="true"
112.                    android:focusable="true"
113.                    android:src="@drawable/map"
114.                    .
115.                    .
116.                    ./>
117.
118.                <TextView
119.                    android:id="@+id/textView2"
120.                    android:layout_width="241dp"
121.                    android:text="@string/no_direction"
122.                    .
123.                    .
124.                    . />
125.            </android.support.constraint.ConstraintLayout>
126.
127.            <android.support.constraint.ConstraintLayout
128.
129.                android:layout_width="match_parent"
130.                android:layout_height="match_parent">
131.
132.                <TextView
133.                    android:id="@+id/textView6"
134.                    android:layout_width="100dp"
135.                    android:text="@string/preview"
136.                    .
137.                    .
138.                    .
139.                    />
140.
141.            </android.support.constraint.ConstraintLayout>
142.
143.            <android.support.constraint.ConstraintLayout
144.                 android:layout_width="match_parent"
145.                 android:layout_height="match_parent">
146.
147.                 <ImageView
148.                     android:id="@+id/imageView"
149.                     android:layout_width="335dp"
150.                     .
151.                     .
152.                     .
153.                     android:src="@drawable/preview"
154.                     />
155.
156.            </android.support.constraint.ConstraintLayout>
157.
158.        </LinearLayout>
159.    </ScrollView>
160.
161. </android.support.constraint.ConstraintLayout>
```

Figure 14. Tour fragment layout XML file

Figure 15. Tour fragment layout

5.4.2 Functionality

Figure 16 shows the code for this fragment.

Purpose:

The Tour fragment forms the second tab.

Implemented Interface(s) (line 1):

This class implements the View.OnClickListener interface.

OnCreateView (Lines 5 - 18):

This function gets called to perform necessary initialization for the tour tab. In particular, it determines that when any of the four button are clicked, the onClick function will be called.

OnStart (Lines 20 - 35):

The onStart function gets called whenever the user clicks the tour tab. It does two jobs: a) it stops the TTS if it is already playing. b) it determines whether the first button should be described by “TAKE ME TO THE FIRST SITE” or “RESUME TOUR” depending on whether any of the stops are already visited by the user.

onClick (Lines 37 - 70):

The onClick function sets the necessary *state variables* and performs the necessary *actioni.* In particular, a) if the "take me to the first site" or "resume" buttons are clicked, the map page will open accordingly; b) if the "start from the nearest stop" button is pressed, the map page will open accordingly; c) if the "reset walking history" button is pressed, the walking history will be erased; and d) if the "listen to stories without going on tour" button is pressed, the map without any directions will be opened.

```
1.  public class tab2 extends Fragment implements View.OnClickListener{
2.
3.      View rootView;
4.
5.      public View onCreateView(LayoutInflater inflater, ViewGroup container,
6.                               Bundle savedInstanceState) {
7.          View rootView = inflater.inflate(R.layout.tab2, container, false);
8.          FloatingActionButton mapShow = (FloatingActionButton) rootView.findViewById(R
    .id.mapShow);
9.          FloatingActionButton startingDir = (FloatingActionButton) rootView.findViewBy
    Id(R.id.startingDir);
10.         FloatingActionButton closestDir = (FloatingActionButton) rootView.findViewByI
    d(R.id.closestDir);
11.         FloatingActionButton resetTour = (FloatingActionButton) rootView.findViewById
    (R.id.resetTour);
12.         mapShow.setOnClickListener(this);
13.         startingDir.setOnClickListener(this);
14.         closestDir.setOnClickListener(this);
15.         resetTour.setOnClickListener(this);
16.         this.rootView = rootView;
17.         return rootView;
18.     }
19.
20.     @Override
21.     public void onStart() {
22.         super.onStart();
23.         MyTTS.stop();
24.
25.         if (MainActivity.seenStops.size()>0){
26.             //We're in the middle of a tour
27.             TextView textView  = rootView.findViewById(R.id.textView3);
28.             textView.setText("RESUME THE TOUR");
29.         }
30.         else{
31.             TextView textView  = rootView.findViewById(R.id.textView3);
32.             textView.setText("TAKE ME TO THE FIRST SITE");
33.         }
34.
35.     }
36.
37.     @Override
38.     public void onClick(View rootView) {
39.
40.         MainActivity.showLandingPage  = false;
41.
42.         switch (rootView.getId()) {
43.             case R.id.mapShow:
44.                 MainActivity.shouldCoverByMap = true;
45.                 MainActivity.onlyShowMarkers = true;
```

```
46.                 MainActivity.uniqueMainActivity.onStart();
47.                 break;
48.             case R.id.startingDir:
49.                 // go to the first site
50.                 MainActivity.shouldCoverByMap = true;
51.                 MainActivity.goToClosetStop = false;
52.                 MainActivity.requestedStop = null;
53.                 MainActivity.onlyShowMarkers = false;
54.                 MainActivity.uniqueMainActivity.onStart();
55.                 break;
56.             case R.id.closestDir:
57.                 // go to the closest site
58.                 MainActivity.shouldCoverByMap = true;
59.                 MainActivity.goToClosetStop = true;
60.                 MainActivity.onlyShowMarkers = false;
61.                 MainActivity.uniqueMainActivity.onStart();
62.                 break;
63.             case R.id.resetTour:
64.                 // resetting
65.                 MainActivity.reset();
66.                 MapsActivityCurrentPlace.reset();
67.                 this.onStart();
68.                 break;
69.         }
70.     }
71. }
```

Figure 16. Tour fragment class

5.4.3 Content

The tour tab is where the user can actually start the tour by pressing one of the buttons. Also, it has an image, which gives a preview of the tour's stops names and their distance from each other (Figure 17).

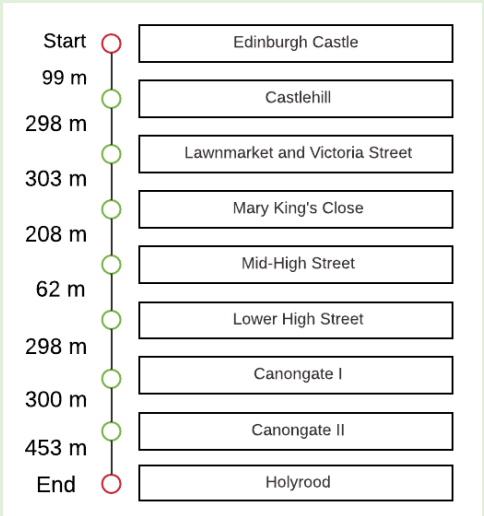


Figure 17. Tour stop's preview

5.5 History Tab

This tab is implemented similar to the Introduction tab, except that it has different content. Figure 18 shows the layout of this tab.

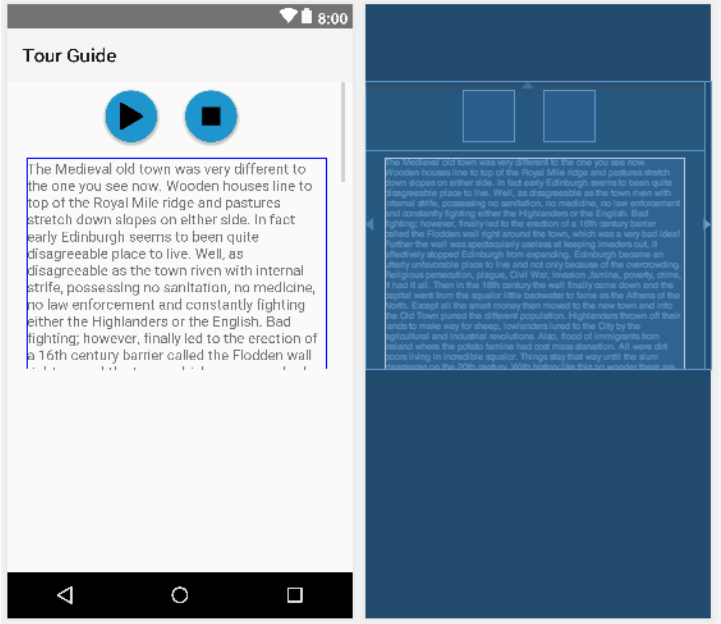


Figure 18. History fragment layout

5.5.1 Content

Information on this tab is not necessary to get the user started the tour, but help the user to know how different the old town was at the middle ages.

Content of the History Tab:

The Medieval old town was very different to the one you see now. Wooden houses line to top of the Royal Mile ridge and pastures stretch down slopes on either side. In fact early Edinburgh seems to been quite disagreeable place to live. Well, as disagreeable as the town riven with internal strife, possessing no sanitation, no medicine, no law enforcement and constantly fighting either the Highlanders or the English. Bad fighting; however, finally led to the erection of a 16th century barrier called the Flodden wall right around the town, which was a very bad idea! Further the wall was spectacularly useless at keeping invaders out, it effectively stopped Edinburgh from expanding. Edinburgh became an utterly unfavorable place to live and not only because of the overcrowding. Religious persecution, plague, Civil War, invasion ,famine, poverty, crime, it had it all. Then in the 18th century the wall finally came down and the capital went from the squalor little backwater to fame as the Athens of the North. Except all the smart money then moved to the new town and into the Old Town purred the different population. Highlanders thrown off their lands to make way for sheep, lowlanders lured to the City by the agricultural and industrial revolutions. Also, flood of immigrants from

Ireland where the potato famine had cost mass starvation. All were dirt poor living in incredible squalor. Things stay that way until the slum clearances on the 20th century. With history like this no wonder there are so many ghosts around, but I'll let you find that out for yourself.

6 TourGuideMap Activity

Purpose:

This activity shows the map to the user. The map is based on Google Maps and shows user's current location, markers of the (ghost) tour, and necessary direction information. Because the user interface of this activity is just a default Google map which is taken from a code base (see below), it is not discussed in technical repot. Figure 19 shows the code for this activity.

Code Base:

The code base of this activity is the CurrentPlaceDetailsOnMap project,[9] one of the android samples tutorials.[10] The project shows a Google map, with one marker on Sydney, Australia. The user can get a direction from his/her current location to the marker's location. The direction can be shown on a google map application of his/her device (not the current google map page).

In order to add direction on the app's map page, another code base was merged with the above code.[11] The codebase is accompanied by a YouTube tutorial,[12] demonstrating how to get direction between two points and draw them on a map. The code was originally designed for an Uber application, but was tailored to Tour Guide's requirements.

Implemented Interface(s) (line 2):

This class implements the following interfaces:

- *OnMapReadyCallback*: necessary function that will be called when the map is loaded and ready (*onMapReady*)
- *RoutingListener*: necessary functions that will be called for routing, including when it succeeds (*onRoutingSuccess*) or fails (*onRoutingFailure*)
- *GoogleMap.OnMarkerClickListener*: necessary functions that will be called when a marker is clicked. This is specified using functions of *InfoWindowAdapter*.

Variables (lines 8 - 27):
uniqueTourGuideMap is the only instance of *TourGuideMap* at any given time. We keep track of that only instance which gets used in the *MainActivity.*

GoogleMap (mMap) is an instance of Android's Google map.

mFusedLocationProviderClient is used to get the current location of the device.

[9] https://github.com/googlemaps/android-samples/tree/master/tutorials/CurrentPlaceDetailsOnMap
[10] https://github.com/googlemaps/android-samples/tree/master/tutorials
[11] https://github.com/jd-alexander/Google-Directions-Android
[12] https://www.youtube.com/watch?v=fZIZ81fn7b4&t=599s

mLastKnownLocation specifies the last known *Location* of the user.

polylines store the route(s) after a direction is found. It can store multiple routes, but the app always queries for one route and stores only the last route.

distancesToRoute stores the last *K=3* distances of the user to the suggested route. This is used to detect when a user is lost.

lastReroutedStop is the last stop (if any) that the user has deviated from its route. This variable is useful because the app gives only one "wrong direction" notification.

locationManager is used to query the user's location periodically using the *locationListener*. The app needs to track the user's location to detect when s(he) has deviated from the specified route.

onCreate function (lines 29 - 47):

This function initializes the above variables which leads to having an active map and *locationListener*.

onStart function (lines 49 - 56):

Whenever the *TourGuideMap* activity gets (re-)loaded, all markers will be pinned on it with the right color (green or red).

pinAllMarkers function (lines 58 - 66):

This function first removes any marker on the map by calling *removeAllMarkers* function (lines 68 - 72). Then it adds all markers (one marker per each geofence in *ED_LANDMARKS*).

pinMarker function (lines 74 – 88):

This function is used to pin one marker on the map. The marker is specified by its name and position. Its icon will be set based on whether the user has visited that marker before (green color) or not (red color).

shouldSuggestRoute function (lines 90 - 95):

It returns a boolean variable which specifies if a route should be suggested to the user, i.e., the tour is still not finished or the user has specifically asked for a direction to some stop.

getRouteToMarker function (lines 98 - 113):

It gets two positions (*LatLong* objects) and finds the route between them.

onMapReady function (lines 115 - 158)

This function is called when the map object is ready. It stores the map object in *mMap*. Then it specifies what actions should be performed when the user clicks on a Marker.

In particular, an object of type *GoogleMap.InfoWindowAdapter* is used and its *getInfoContents* function is implemented, which does the following: It sets the title and snippet of the marker, which will be shown when it is clicked. It also specifies when the title of the marker is pressed, a story activity for that marker should be launched.

Finally, the *getDeviceLocation* and *pinAllMarkders* functions are called.

getDeviceLocation function (lines 160 – 210):

The function gets the device's location and when it's computed it does the following: It first moves the camera to that position. If the user has asked to go to the closest stop, that stop is found using *findClosestStop* function. Finally, a route from the user's current location to the desired stop (closest stop, next stop, or requested stop by user) is drawn.

getNextStop function (lines 212 - 217):

This function finds the next unvisited stop in the natural order. If the current stop is the last stop and the user has not visited all the stops, the first unvisited stop will be returned. The implementation details of this function is removed for brevity.

setLocationManager (lines 219 - 275):

This function is used for helping the user cope with getting lost. It creates a *locationManager* that asks for user's location every 7 seconds and/or every10 meters that the user moves. The *locationManater* gets as input a *locationListener*, which specifies the actions that should be done every time the user's location is found by implementing *onLocationChanged* function (lines 224 - 267).

The *onLocationChanged* calculates distance of the user to the current route. Every *K=3* times that the distance is computed, if the minimum of those distances is greater than *C=70*, an alert will be given to the user using TTS (line 252).

onBackPressed function (lines 277 - 282): It stops the TTS (if playing) and opens the *LandingPage* activity by closing the *TourGuideMap* activity and setting *showLandingPage* to true.

reset function (lines 284 - 286):

This function sets the *lastReroutedStop* variable to empty string.

fisish function (lines 288 - 292):

It makes sure that the locationManager stops tracking the location of the user.

```
1.  public class TourGuideMap extends AppCompatActivity
2.          implements OnMapReadyCallback ,RoutingListener, GoogleMap.OnMarkerClickListen
    er{
3.
4.      .
5.      .
6.      .
7.
8.      public static TourGuideMap uniqueTourGuideMap;
9.      private GoogleMap mMap;
10.
11.     // The entry point to the Fused Location Provider.
12.     private FusedLocationProviderClient mFusedLocationProviderClient;
13.
14.     // The geographical location where the device is currently located. That is, the
    last-known
15.     // location retrieved by the Fused Location Provider.
```

```
16.     private Location mLastKnownLocation;
17.
18.     //Stores the route(s). In Tour Guide app, we only keep one route.
19.     private List<Polyline> polylines = new ArrayList<>();
20.
21.     //used for coping with getting lost
22.     List<Double> distancesToRoute = new ArrayList<>();
23.     public static String lastReroutedStop = "";
24.     LocationManager locationManager;
25.     LocationListener locationListener;
26.
27.     List<Marker> pinnedMarkers = new ArrayList<>();
28.
29.     @Override
30.     protected void onCreate(Bundle savedInstanceState) {
31.         .
32.         .
33.         .
34.
35.         uniqueTourGuideMap = this;
36.         polylines = new ArrayList<>();
37.
38.         // Construct a FusedLocationProviderClient.
39.         mFusedLocationProviderClient = LocationServices.getFusedLocationProviderClien
    t(this);
40.
41.         // Build the map.
42.         SupportMapFragment mapFragment = (SupportMapFragment) getSupportFragmentManag
    er()
43.                 .findFragmentById(R.id.map);
44.
45.         mapFragment.getMapAsync(this);
46.         setLocationManager();
47.     }
48.
49.     @Override
50.     public void onStart() {
51.         super.onStart();
52.         if (mMap!=null){
53.             pinAllMarkers();
54.         }
55.
56.     }
57.
58.     public void pinAllMarkers () {
59.         removeAllMarkers();
60.         pinnedMarkers = new ArrayList<>();
61.         int i = 0;
62.         for (String placeName: Constants.Ed_LANDMARKS.keySet()){
63.             pinMarker(Constants.Ed_LANDMARKS.get(placeName), placeName, i);
64.             i++;
65.         }
66.     }
67.
68.     private void removeAllMarkers(){
69.         for (Marker m: pinnedMarkders){
70.             m.remove();
71.         }
72.     }
73.
74.     public void pinMarker (LatLng latlong, String placeName, int i) {
```

```
75.         Marker marker = mMap.addMarker(new MarkerOptions()
76.                 .title(placeName)
77.                 .position(latlong));
78.
79.         pinnedMarkers.add(marker);
80.
81.         if (MainActivity.seenStops.contains(placeName) && ! MainActivity.onlyShowMark
    ers) {
82.
83.             marker.setIcon(BitmapDescriptorFactory.fromResource(Constants.seenStop2Si
    gn.get(i)));
84.         } else {
85.
86.             marker.setIcon(BitmapDescriptorFactory.fromResource(Constants.unSeenStop2
    Sign.get(i)));
87.         }
88.     }
89.
90.     static boolean shouldSuggestRoute(){
91.         if (MainActivity.seenStops.size()==Constants.Ed_LANDMARKS.size() && MainActiv
    ity.requestedStop==null){
92.             return false;
93.         }
94.         return true;
95.     }
96.
97.
98.     private void getRouteToMarker(LatLng latLng1, LatLng latLng2) {
99.         if (MainActivity.onlyShowMarkers){
100.                    return;
101.                }
102.                else if (shouldSuggestRoute() && mLastKnownLocation!=null){
103.                    distancesToRoute = new ArrayList<>();
104.                    Routing routing = new Routing.Builder()
105.                            .travelMode(AbstractRouting.TravelMode.WALKING)
106.                            .withListener(this)
107.                            .alternativeRoutes(false)
108.                            .waypoints( latLng1, latLng2)
109.                            .key(MY_API_KEY)
110.                            .build();
111.                    routing.execute();
112.                }
113.            }
114.
115.            /**
116.             * Manipulates the map when it's available.
117.             * This callback is triggered when the map is ready to be used.
118.             */
119.            @Override
120.            public void onMapReady(GoogleMap map) {
121.                mMap = map;
122.                mMap.setOnMarkerClickListener(this);
123.                final Context thisMapActivity = this;
124.
125.                //what the map should show when clicked on markers
126.                mMap.setInfoWindowAdapter(new GoogleMap.InfoWindowAdapter() {
127.                    @Override
128.                    // Return null here, so that getInfoContents() is called next.
129.                    public View getInfoWindow(Marker arg0) {
130.                        return null;
131.                    }
```

```
132.                    @Override
133.                    public View getInfoContents(Marker marker) {
134.                        // Inflate the layouts for the info window, title and snippet
    .
135.                        View infoWindow = getLayoutInflater().inflate(R.layout.custom
    _info_contents,
136.                                (FrameLayout) findViewById(R.id.map), false);
137.
138.                        TextView title = ((TextView) infoWindow.findViewById(R.id.tit
    le));
139.                        title.setText(marker.getTitle());
140.
141.                        TextView snippet = ((TextView) infoWindow.findViewById(R.id.s
    nippet));
142.                        snippet.setText(marker.getSnippet());
143.
144.                        mMap.setOnInfoWindowClickListener(new GoogleMap.OnInfoWindowC
    lickListener() {
145.                            public void onInfoWindowClick(Marker marker)
146.                            {
147.                                MainActivity.activeStoryStopName = marker.getTitle();

148.                                MainActivity.showStory(thisMapActivity);
149.                            }
150.                        });
151.                        return infoWindow;
152.                    }
153.
154.                });
155.
156.                getDeviceLocation(true);
157.                pinAllMarkers();
158.            }
159.
160.            /**
161.             * Gets the current location of the device, and positions the map's camer
    a. Also, gets direction
162.             * to the next stop
163.             */
164.            public void getDeviceLocation(final boolean shouldMoveCamera) {
165.                /*
166.                 * Get the best and most recent location of the device, which may be
    null in rare
167.                 * cases when a location is not available.
168.                 */
169.                try {
170.                    if (mLocationPermissionGranted) {
171.                        Task<Location> locationResult = mFusedLocationProviderClient.
    getLastLocation();
172.                        locationResult.addOnCompleteListener(this, new OnCompleteList
    ener<Location>() {
173.                            @Override
174.                            public void onComplete(@NonNull Task<Location> task) {
175.
176.                                if (task.isSuccessful()) {
177.                                    // Set the map's camera position to the current l
    ocation of the device.
178.                                    mLastKnownLocation = task.getResult();
179.                                    if (shouldMoveCamera){
180.                                        mMap.moveCamera(CameraUpdateFactory.newLatLng
    Zoom(
```

```
                                                new LatLng(mLastKnownLocation.getLati
    tude(),
                                                        mLastKnownLocation.getLongitu
    de()), 15));
                                    }

                                    if (MainActivity.goToClosetStop){
                                        MainActivity.requestedStop = uniqueTourGuideM
    ap.findClosestStop();
                                        MainActivity.goToClosetStop = false;
                                        MainActivity.nextStopTovisit = uniqueTourGuid
    eMap.getNextStop();
                                    }

                                    String placeName =  MainActivity.nextStopTovisit;

                                    LatLng firstToVisit =  Constants.Ed_LANDMARKS.get
    (placeName);

                                    if (shouldMoveCamera){
                                        getRouteToMarker(new LatLng(mLastKnownLocatio
    n.getLatitude(),
                                                mLastKnownLocation.getLongitude()),fi
    rstToVisit);
                                    }

                                } else {//zoom to default location
                                    .
                                    .
                                    .
                                }
                            }
                        });
                    }
                } catch (SecurityException e)  {
                    Log.e("Exception: %s", e.getMessage());
                }
            }

            //get the first unvisited stop
            public static String getNextStop(){
                .
                .
                .
            }

            void setLocationManager(){
                locationManager = (LocationManager) getSystemService(LOCATION_SERVICE
    );
                try {
                    locationListener = new LocationListener() {
                        @Override
                        public void onLocationChanged(Location location){
                            try{
                                if (polylines.size()>0) {
                                    List<LatLng> latLongs = (polylines.get(0).getPoin
    ts());
                                    double distanceFromPolyLine = bdccGeoDistanceAlgo
    rithm.
                                            my_bdccGeoDistanceFromPolyLine(latLongs,
    new LatLng(location.
```

```
                                                    getLatitude(), location.getLongit
    ude()));

                                    distancesToRoute.add(distanceFromPolyLine);

                                    if (distancesToRoute.size() < Constants.NUMWRONGD
    IRLIST){
                                        return;
                                    }

                                    double minDistance = 100000 ;

                                    //now it's time to get the min
                                    for (double distance: distancesToRoute) {
                                        if (distance<minDistance){
                                            minDistance = distance;
                                        }
                                    }
                                    distancesToRoute = new ArrayList<>();

                                    if (minDistance > Constants.MINWRONGDIST){
                                        if (!lastReroutedStop.equals(MainActivity.nex
    tStopTovisit)){
                                            lastReroutedStop = MainActivity.nextStopT
    ovisit;
                                            getDeviceLocation(true);
                                            MyTTS.play("Wrong direction. Please check
     the map again.",false);
                                        }
                                    }
                                }
                            }
                            catch (Exception e){
                                .
                                .
                                .
                            }
                        }
                        .
                                .
                                .

                    };
                    //How often check for getting lost?
                    locationManager.requestLocationUpdates(LocationManager.GPS_PROVID
    ER, 7000, 10, locationListener);
                } catch (SecurityException e) {
                    .
                    .
                    .
                }
            }

            @Override
            public void onBackPressed(){
                MyTTS.stop();
                MainActivity.showLandingPage  = true;
                super.onBackPressed();
            }

            static void reset(){
```

```
285.                lastReroutedStop = "";
286.            }
287.
288.            @Override
289.            public void finish() {
290.                super.finish();
291.                locationManager.removeUpdates(locationListener);
292.            }
293.        }
```

Figure 19. TourGuideMap activity class

7 Story Activity

Purpose:

This activity forms the story page of the app. It consists of three buttons (Play, Stop, Direction) and a text view. For each stop, the text view shows the story of that stop and the buttons provide the necessary functionalities so that the user can play and stop the story, also get direction to that stop (or its next stop).

Code Base:

I added a new AppCompatActivity to the main project.

7.1 User Interface

The user interface of the story activity is determined by *story_activity.xml* (Figure 20). Figure 21 shows its layout graphically. The main layout is a *ConstraintLayout* and has the following components.

High level properties (lines 2 - 6):

The high-level properties of the layout such as its size and position are specified.

ImageView (lines 8 - 15):

The *ImageView* (*stopImage*) defines the position and the id of the image that is placed at the top of the story page. It has a default image from the Edinburgh castle, but the image gets loaded dynamically based on the stops in the story activity code.

ConstraintLayout (lines 18 - 55):

This is used to build a container and put the three buttons in it. Each button is defined using a *FloatingActionButton* (lines 25 – 33 for  , lines 35 – 43 for  , lines 45 – 53 for ) . In these lines the exact position of the buttons, their id, their shape, and the functions to be called when they are clicked are specified.

ScrollView (lines 57 - 79):

It makes the content at the bottom of the story page scrollable. It consists of a *LinearLayout* that has a *TextView* (the story for each stop).

```
1.  <?xml version="1.0" encoding="utf-8"?>
2.  <android.support.constraint.ConstraintLayout
3.      .
4.      .
5.      .
6.      tools:context="Story">
7.
8.      <ImageView
9.          android:id="@+id/stopImage"
10.         android:layout_width="397dp"
11.         .
12.         .
13.         .
14.         android:contentDescription="@string/app_name"
15.         app:srcCompat="@drawable/ghost_tour" />
16.
17.
18.     <android.support.constraint.ConstraintLayout
19.         android:id="@+id/constraintLayout2"
20.         android:layout_width="313dp"
21.         .
22.         .
23.         .>
24.
25.         <android.support.design.widget.FloatingActionButton
26.             android:id="@+id/play"
27.             android:layout_width="64dp"
28.             .
29.             .
30.             .
31.             android:clickable="true"
32.             android:onClick="playButtonHandler"
33.             android:src="@drawable/play"/>
34.
35.         <android.support.design.widget.FloatingActionButton
36.             android:id="@+id/stop"
37.             android:layout_width="64dp"
38.             .
39.             .
40.             .
41.             android:clickable="true"
42.             android:onClick="stopButtonHandler"
43.             android:src="@drawable/stop"/>
44.
45.         <android.support.design.widget.FloatingActionButton
46.             android:id="@+id/Direction"
47.             android:layout_width="64dp"
48.             .
49.             .
50.             .
51.             android:clickable="true"
52.             android:onClick="directionHandler"
53.             android:src="@drawable/directions"/>
54.
```

```
    </android.support.constraint.ConstraintLayout>

    <ScrollView
        android:id="@+id/textStories"
        android:layout_width="344dp"
        .
        .
        .>

        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:orientation="vertical">

            <TextView
                android:id="@+id/storyText"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_margin="@dimen/text_margin"
                android:textSize="14dp"
                android:gravity="center"
                android:justificationMode="inter_word"
                android:text="" />
        </LinearLayout>
    </ScrollView>

    <TextView
        android:id="@+id/storyName"
        android:layout_width="330dp"
        .
        .
        ./>

</android.support.constraint.ConstraintLayout>
```

Figure 20. Story activity layout XML file

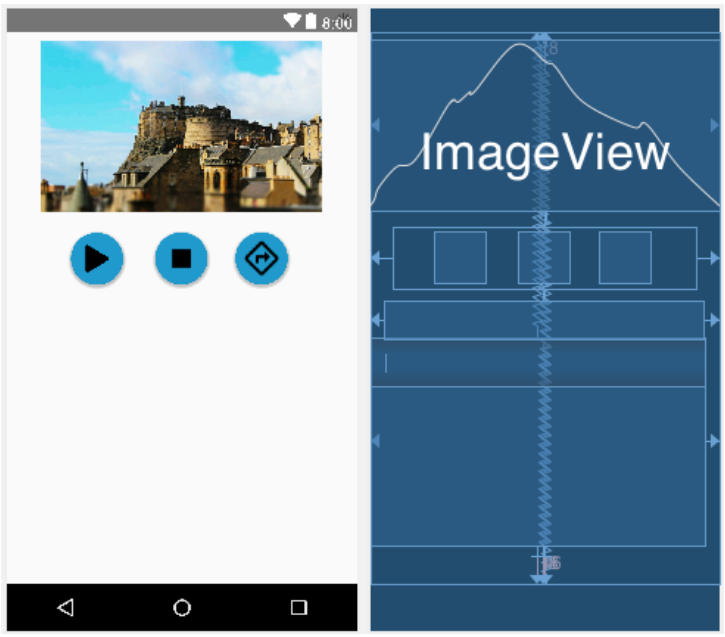


Figure 21. Story activity layout

7.2 Functionality

Implemented Interface(s) (Line 1):

This class implements the *TTSFinishHandler* interface. Any class that implements this interface should specify the necessary actions that need to be performed when the TTS finishes reading. For example: closing the story. Figure 22 shows the code for this class.

Variables (Lines 3 - 9):

uniquestory is the only instance of Story page at any given time. We keep track of that only instance which gets used in the *MainActivity*.

mDirButton specifies the button that when clicked the direction to the stop will be shown on the map.

mPlayButton is the button to play the story.

stopsImage is the image of the story's stop.

enteredGeofence is a boolean variable that helps with getting direction and checks weather the user is inside one of the geofences or not.

preStory is a string to be played before the story starts.

postStory is a string to be played after the story is finished.

onCreate function (lines 11 - 77):
This function specifies when *prestory* and *poststory* should be loaded or be left as an empty string, also how to deal with playing, stopping or getting direction. In particular, it sets the uniqueStory (line 14) and the layout (line 15). It then loads play and direction buttons (lines 18 - 20). In the case that the user wants to listen to story without physically going to on the tour, it disable the functionality of the direction button and fade its color on the story page (lines 21 - 24).

It loads the default *stopImage* which is later on set dynamically (line 26). Then, it specifies when *prestory* and *poststory* should be loaded or be left as an empty string.

- If the user just entered a geofence (and pressed the notification), or s(he)'s already in one stop and asks for its story to be played: fill the content for the prestory.
- If the next stop to be seen is the one exactly after the current stop, fill the content of postStory
- Or if the user had seen more than 50% of the tour and now is in the last geofence, fill the *postStory* of the last stop which congrats the user for finishing the tour (lines 47 - 50)

The code then plays the story automatically (line 58), load and sets the stop's image, stop's story and stop's name (lines 61 - 76).

playButtonHandler (lines 86 - 91):

This function starts playing the stop's story, when the play button on the story page is pressed.

stopButtonHandler(lines 93 - 96):

This function stops playing the stop's story, when the stop button on the story page is pressed.

onBackPressed (lines 98 - 104):

This function is called when the user presses back while in the story page. In that case, this function stops the TTS and opens the map activity with a route showing the direction to the next stop.

directionHandler (lines 106 - 127):

This function closes the story page and shows the direction on the map. It handles showing the direction in the following three cases:

- When the user asks a direction explicitly (line 107 - 113): In this case the direction on the map is to that requested stop.
- When the user visits all the tour's stops (lines 114 - 122): In this case there will be no more direction.
- When the user has seen a stop and wants the direction to see the next (lines 123 - 125): In this case, a direction to the next stop will be shown.

onTTSFinish (lines 129 - 136):

This is the function that closes the story page when TTS gets finished.

```
1.  public class Story extends AppCompatActivity implements TTSFinshHandler{
2.
3.      public static Story uniqueStory;
4.      private FloatingActionButton mDirButton;
5.      public FloatingActionButton mPlayButton;
6.      private ImageView stopsImage;
7.      public boolean enteredGeofence = false;//as opposed to clicked to get direction
8.      String preStory = "";//to be played before the story is started
9.      String postStory = "";//to be played after the story is finished
10.
11.     @Override
12.     protected void onCreate(Bundle savedInstanceState) {
13.         super.onCreate(savedInstanceState);
14.         uniqueStory = this;
15.         setContentView(R.layout.activity_scrolling);
16.         Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
17.         setSupportActionBar(toolbar);
18.         mPlayButton = (FloatingActionButton) findViewById(R.id.play);
19.
20.         mDirButton= findViewById(R.id.Direction);
21.         if (MainActivity.onlyShowMarkers){
22.             mDirButton.setEnabled(false);
23.             mDirButton.setAlpha(0.3f);
24.         }
25.
26.         stopsImage = findViewById(R.id.stopImage);
27.
28.         postStory = "";
29.         preStory = "";
30.
31.         /*
```

```
32.         if just entered a geofence (and pressed the notification), or she's aleady in
     one stop
33.         and asks for story
34.          */
35.         if (MainActivity.justEnteredGeofence || (MainActivity.lastEnteredGeofence!=nu
    ll && MainActivity.lastEnteredGeofence.getRequestId().equals(MainActivity.activeStory
    StopName))){
36.             enteredGeofence = true;
37.             MainActivity.justEnteredGeofence = false;
38.             MainActivity.nextStopTovisit = MapsActivityCurrentPlace.getNextStop();
39.             String stopName = MainActivity.lastEnteredGeofence.getRequestId();
40.             preStory = Constants.id2PreStory.get(stopName);
41.
42.             try{
43.                 if (Constants.id2Dir.containsKey(stopName)){
44.                     int nextIdx = Constants.id2Idx.get(MainActivity.nextStopTovisit);

45.                     int thisIdx = Constants.id2Idx.get(stopName);
46.                     //determines when postStory can be shown.
47.                     if (nextIdx==(thisIdx+1) || (thisIdx==Constants.id2Idx.size()-
    1 && MainActivity.
48.                             seenStops.size()>=.5*Constants.Ed_LANDMARKS.size())){
49.                         postStory = " "+ Constants.id2Dir.get(stopName);
50.                     }
51.                 }
52.             }
53.             catch (Exception e){
54.                 e.printStackTrace();
55.             }
56.
57.             //play automatically
58.             Story.uniqueStory.mPlayButton.performClick();
59.         }
60.
61.         String stopName = MainActivity.activeStoryStopName;
62.         String currentStory = getCurrentStory(stopName);
63.
64.         //load (and set) the stop's image!
65.         Resources resources = getApplicationContext().getResources();
66.         int resourceId = resources.getIdentifier(stopName.toLowerCase(), "drawable",

67.                 getPackageName());
68.         stopsImage.setImageResource(resourceId);
69.
70.         //load (and set) the stop's story!
71.         TextView textView = findViewById(R.id.storyText);
72.         textView.setText(currentStory);
73.
74.         //load (and set) the stop's name!
75.         TextView textView1 = findViewById(R.id.storyName);
76.         textView1.setText(stopName);
77.     }
78.
79.     //gets the (full) story
80.     String getCurrentStory(String stopName){
81.         String currentStory = Constants.id2Story.get(stopName);
82.         currentStory = preStory+currentStory+postStory;
83.         return currentStory;
84.     }
85.
86.     //plays the story
```

```
87.     public void playButtonHandler(View view){
88.         String stopName = MainActivity.activeStoryStopName;
89.         String currentStory = getCurrentStory(stopName);
90.         MyTTS.play(currentStory,true,this);
91.     }
92.
93.     //stops the TTS
94.     public void stopButtonHandler(View view){
95.         MyTTS.stop();
96.     }
97.
98.     //going back to map!
99.     @Override
100.            public void onBackPressed(){
101.                MyTTS.stop();
102.                MainActivity.shouldCoverByMap = true;
103.                super.onBackPressed();
104.            }
105.
106.            public void directionHandler(View view){
107.                //This is when user asks explicitly to go to some stop.
108.                if (!enteredGeofence){
109.                    MainActivity.requestedStop = MainActivity.nextStopTovisit = MainA
    ctivity.activeStoryStopName;
110.                }
111.
112.                MapsActivityCurrentPlace.uniqueMapsActivityCurrentPlace.getDeviceLoca
    tion(true);
113.
114.                //This is when the user has seen everything and the tour has finished
    !
115.                if (!MapsActivityCurrentPlace.shouldSuggestRoute()){
116.                    MainActivity.showLandingPage  = true;
117.                    finish();
118.                    if (MapsActivityCurrentPlace.uniqueMapsActivityCurrentPlace!=null
    ){
119.                        MapsActivityCurrentPlace.uniqueMapsActivityCurrentPlace.finis
    h();
120.                    }
121.
122.                }
123.                else{//normal case: go back to map and show the direction
124.                    onBackPressed();
125.                }
126.
127.            }
128.
129.            // closes the story page when TTS gets finished!
130.            @Override
131.            public void onTTSFinish() {
132.                if (!MapsActivityCurrentPlace.shouldSuggestRoute()){
133.                    MainActivity.showLandingPage  = true;
134.                }
135.                finish();
136.            }
137.        }
```

Figure 22. Story activity class

8 Content

The content for the stories of each site is mainly taken from the "Edinburgh: City of the Dead" book and the "Haunted Edinburgh" app. The writer of the book - J. A. Henderson - is the narrator in the app and based on what he says in the app he has at least 30 years' experience in tour guiding. The collected format content was either in vocal or textual format. The voice was converted to text using an online speech to text convertor.[13] The other apps that their contents are used in the app are "Rick Steve's Europe audio tour" and "Voice Map". Figure 23 shows the cover page of the book and Figure 24 shows the icon of these three apps. The story for each stop is available in Appendix I.

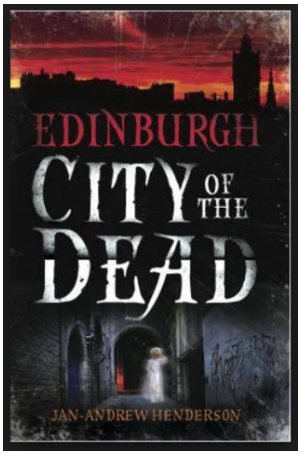


Figure 23. The Cover page of the "Edinburgh: City of the Dead" Book

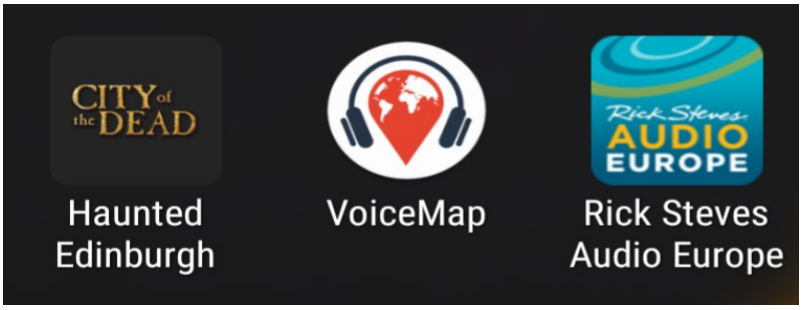


Figure 24. The Icons of the Three Android Apps that their Story Content have been Used in this Project

9 User Testing

This section shows the document and the questionnaire used in user testing. Users were asked questions graded on the Likert Scale (from negative to positive), also multiple-choice questions. The raw data from the user testing is available in Appendix II.

Storytelling in Space and Time:

[13] https://speechnotes.co/

A Ubiquitous Mobile Application

Laleh Moussavi

Introduction

Thank you for taking part in the evaluation of the "Tour Guide" application. This android application is a walking audio tour and has been developed as part of a master's dissertation at the university of Edinburgh. The mobile application uses your location data to help you explore downtown Edinburgh by using an embedded map along with an audio guide. Please enable your data and GPS of your phone during the tour. At the end of your experience with the mobile application you will be asked to fill out a questionnaire. The whole evaluation will take about one to two hour(s).

How to work with the mobile application?

The mobile application offers a ghost tour of the royal mile, which has 9 sites. Each of the tour sites has its own title and track number on the map. It is best to just follow the tour in the order that is laid out. But you can skip ahead or tailor your itinerary to your taste and time. When you get close to a site, you will get a notification. If you click on it the related story will play automatically. As you leave each site you will get some guidance on how to find your next site.

Ethics and Safety

Please note that your safety is our first priority. Please watch for cars and buses when crossing streets, while using your phone. Also, to comply with the university's ethics, users must be over 18. The results from this survey will only be anonymously incorporated into the dissertation. Thank you again and enjoy the tour!

At the conclusion of this experiment, please complete the following questionnaire.

1.Your age: ……..

2. Your gender: ……...…….

3. Your Occupation: ……..…….

4. Have you been on a human walking tour before?

Yes No

5. Have you used any mobile walking tour app before?

Yes No

6. I prefer a mobile app tour rather than a human tour (please consider the fee payment & time flexibility when answering to this question)

Strongly Disagree Disagree Neutral Agree Strongly Agree

7. It was fun to use the mobile app

Strongly Disagree Disagree Neutral Agree Strongly Agree

8. I enjoyed the layout of the mobile app

Strongly Disagree Disagree Neutral Agree Strongly Agree

9. I found it is easy to navigate using the mobile app

Strongly Disagree Disagree Neutral Agree Strongly Agree

10. I found the automatic text to speech convertor working properly

Strongly Disagree Disagree Neutral Agree Strongly Agree

11. I enjoyed the contents of the app

Strongly Disagree Disagree Neutral Agree Strongly Agree

12. Notifications were accurate and triggered at the right time

Strongly Disagree Disagree Neutral Agree Strongly Agree

13. Notifications made exploring the city easier

Strongly Disagree Disagree Neutral Agree Strongly Agree

14. Which of the following did you find more convenient in guiding you through the tour

The directions on the map
The situational awareness direction information suggested at the end of each story (for example, "carry on down the hill")

15. Which of the issues below was the biggest problem during your experience with the mobile app?

I experienced bugs (the app did not work as I one might expect)
The app was visually unappealing
The app was missing features I needed
The app crashed
The app was confusing to use
Other (please specify)

16. What do you like most about the mobile app?

Look and feel
Functionality
Stability
Speed
Navigation
Content
Other (please specify)

17. What improvements/changes would you like to see in the mobile app?

18. How would you rate the mobile app?

1 2 3 4 5

10 References

ANDROIDDEVELOPERS. 2018a. *Android Studio* [Online]. Developers. Available: https://developer.android.com/studio/ [Accessed].

ANDROIDDEVELOPERS. 2018b. *App Manifest Overview* [Online]. Developers. Available: https://developer.android.com/guide/topics/manifest/manifest-intro [Accessed].

ANDROIDDEVELOPERS. 2018c. *Configure your build* [Online]. Developers. Available: https://developer.android.com/studio/build/ [Accessed].

ANDROIDDEVELOPERS. 2018d. *Developer workflow basics* [Online]. Available: https://developer.android.com/studio/workflow [Accessed].

ANDROIDDEVELOPERS. 2018e. *Fragments* [Online]. Available: https://developer.android.com/guide/components/fragments [Accessed].

ANDROIDDEVELOPERS. 2018f. *Google Play Developer API* [Online]. Developers. Available: https://developer.android.com/google/play/developer-api [Accessed].

ANDROIDDEVELOPERS. 2018g. *Meet Android Studio* [Online]. Available: https://developer.android.com/studio/intro/ [Accessed].

ANDROIDDEVELOPERS. 2018h. *Projects overview* [Online]. Available: https://developer.android.com/studio/projects/ [Accessed].

JOEY. 2010. *SDK vs IDE* [Online]. stackoverflow. Available: https://stackoverflow.com/questions/3196986/difference-between-sdk-and-ide [Accessed].

11 Appendix I: Content of the Story for Each Stop

Edinburgh Castle:

Our first stop is Edinburgh Castle at the top of the mile. You can stand outside and admire it's towering walls, so many attackers have done. Or you can pay the rather considerable entrance fee and explore the place. It's worth it though because the castle is amongst the Edinburgh's must visit sites. Containing such treasures as: the Scottish Crown Jewels, the giant Mons Meg, a splendid War Memorial and Edinburgh's oldest building, the 12th century St Margaret's Chapel. **(Pre-Story)**
The castle is haunted by several apparitions, hardly surprising given its history of conflict. It is haunted by Lady Janet Douglas, burned as a witch in 1537, on the trumped up charge of trying to assassinate James IV. Also, the ghost of John Graham of Claverhouse first appeared in 1689 after he was killed at the Battle of Killiekrankie. The dungeons are haunted by former prisoners of war, especially one who hid in a barrow of dung, hoping to be carried outside. He was tipped over the side of the ramparts instead and now tries to push unsuspecting tourists off in revenge. Hasn't managed yet, but you might be the first. Then there's an invisible drummer who only plays when the castle is about to be attacked. So if you hear him, start running. And look out for a phantom piper and a spooky steward, murdered by the castle governor in the 17th century. **(Story)**
Our next stop is Castlehill, which starts at The Esplanade, right outside the castle. **(Post story)**

Castlehill:

If you didn't fancy paying the entrance fee you're already standing on the Castlehill. Since the other three sides of the fortification overlook sheer Cliffs, this was the only feasible direction of attack which left potential Invaders horribly exposed. it is said that the length of the approach was exactly the distance an arrow could be accurately fired. The Esplanade was paved in 1753 and is now used for the Edinburgh Military Tattoo each August. **(Pre-Story)**
The sounds of a phantom army can sometimes be heard marching along the esplanade towards the castle. Or perhaps it's the guy next to you eating crisps! At the bottom of the esplanade is a plaque and well, commemorating the spot where over 300 innocent women were burned as witches. In the area around Ramsay Garden you can sometimes see a hunchback in 18th century dress, dragging a wooden trunk. Further down Castlehill, the Whisky Heritage Centre has the ghost of a former master blender, which gives you a chance to experience a different kind of spirit. Sorry, couldn't resist that. The next door Witchery Restaurant's name says it all really. Filled with Gothic finery it is haunted, unsurprisingly, by one of the women burned as a witch on Castlehill. **(Story)**
Walk a few yards further and you'll come to the gothic spire of the Hub, headquarters of the Edinburgh International Festival. This is where Castlehill ends and lawnmarket begins. Cross over the roundabout and take a quick detour down the upper Bow to find yourself on a balcony overlooking Victoria Street. This was once the West Bow and is Notorious for supernatural sightings and dark deeds. **(Post story)**

Lawnmarket and Victoria Street:

Just off Lawnmarket is Victoria Street, which has a very dark past. Top villain is 'The Wizard of the West Bow', Major Thomas Weir, executed for witchcraft in 1670. Weir's house, reputed to be the most haunted in Scotland, was discovered to be part of the Quaker Meeting house on Victoria Terrace. A phantom coach, pulled by headless horses and ridden by the devil himself, gallops down the street late at night — presumably giving the ghost of Major Weir a lift. A sailor named Angus Roy, crippled on a voyage in 1820, is also seen dragging his leg. Shame he picked such a steep street to hang out. Brodie's Close on

Lawnmarket is, quite naturally, haunted by the notorious baddie Deacon Brodie - inspiration for Dr Jekyll and Mr Hyde. Round the corner, in vaults under George IV Bridge (once used as a prison), lurks an unidentified highland chief in manacles. **(Story)**
Head back to the royal mile and Walk a few yards further down hill, shortly after the West Parliament Square, across the street, you will see a visitor attraction called Mary King's Close. This is where our next stop would be. **(Post story)**

Mary King's Close:

According to legend, plague broke out in 1645 and the infected inhabitants were walled up inside. It's a chilling story but, happily, not true. The residents were actually quarantined outside the city and the close was inhabited until the last resident left in 1904. The lower half of the Close was eventually demolished, but the upper half was covered over and is now a spectacular underground visitor attraction. It got its creepy reputation because of a 1685 bestseller by Professor George Sinclair called Satan's Invisible World, which described the close as a supernatural hotbed. Haunted these days by a little girl called Annie. Visitors bring her dolls and now there's a whole room full of them. This is either touching or immensely creepy, depending on how much you like dolls and kids. **(Story)**
On the other side of the street you can see the Parliament Square. It is said that this is where the devil appeared in 1513, reading out a list of those who would die in the disastrous Battle of Flodden - including King James IV. Now head down the hill and our next haunted stop would be the Bell's Wynd, about 200 meters from here, located in the Mid-High Street. **(Post story)**

Mid High Street:

Bell's Wynd is haunted by a Mrs Guthrie. Murdered by her husband for having an affair, her body remained undiscovered in a locked room for two decades, in Europe's most overcrowded street. Hmmmm. In the 19th century a secret tunnel was found in the dungeons of Edinburgh Castle. It was too small to accommodate an adult, so the town council sent in a drummer boy and follow the sound down the mile until it stopped under the Kirk. Figuring it was a lost cause, they sealed out the tunnel and now drumming can be head under the building. Also, the Scotsman Hotel on the North Bridge had a bunch of ghosts, though they aren't often seen these days. Either you can't believe everything you read in the press or the present hotel is too expensive for them. The haunted Radisson Blu Hotel, built over the house of George Mackenzie, was damaged in one of the area's many fires, with the roof burning off in 1987. According to legend, John Spottiswood, Bishop of St Andrews (1565-1639) haunts the Mitre Bar. This seems like a very sensible place to spend eternity. **(Story)**
Cross the road and a few yards further down you will come to the Canongate Tolbooth. Right its next door is Canongate Kirkyard which has one little tail well worth recounting, Ebenezer Scrooge. **(Post story)**

Lower High Street:

The laundry room of the hostel in the Blackfriars street has a paranormal presence and the thoroughfare has a couple of pretty classy spooks. **(Pre-Story)**
Tyrannical Cardinal David Beaton and debauched Lord Darnley, husband of Mary Queen of Scots are amongst the Blackfriars Street's rather impressive ghosts. In 1567, Darnley was chased and strangled by assassins and now his frantic footsteps can be heard at night. According to folklore, a group of children were locked inside a local nursery after one was discovered to have plague symptoms. The indignant mothers ran down the Mile and forced their way past the guards - so they were locked in too. The sound of children crying can now be heard in the Museum of Childhood, which is already creepy enough, as it's

filled with Victorian dolls. The World's End Bar was the site of an infamous double homicide in 1977 and St Mary's Street is haunted by a young woman, murdered in 1916. **(Story)**
Continue down the hill and once you have passed St Mary's street you are in the Canongate, which was once a separate burgh outside the city walls. **(Post story)**

Canongate I:

Look out for brass plaques in the road, which mark the boundary line. This was where the remains of yet more executed criminals were displayed on spikes which didn't exactly say welcome to Edinburgh. **(Pre-Story)**
Three very different spirits stalk the Canongate. The first is 'Daft Jamie' Wilson, a simpleton killed by William Burke and William Hare (they murdered another, Mary Paterson, who haunts Gullan's Close). Altogether Burke and Hare suffocated sixteen people, selling the bodies to Dr Robert Knox at Edinburgh University's Anatomy department for his students to dissect. The second is a hooded figure, who may be the 17 th century minister and wife killer, John Kello. Then there's a burning woman, daughter of a well to do family, who got pregnant by a servant and was killed in an 'accidental' file. On the balcony of Moray House, the Covenanter leader the Marquess of Argyll spat on his arch enemy, the Marguess of Montrose as he was being led to his execution. Argle suffered the same fate shortly afterwards. **(Story)**
The next stop is Lower High street. To go to there, carry on downhill until you reach Blackfriars Street. **(Poststory)**

Cannongate II:

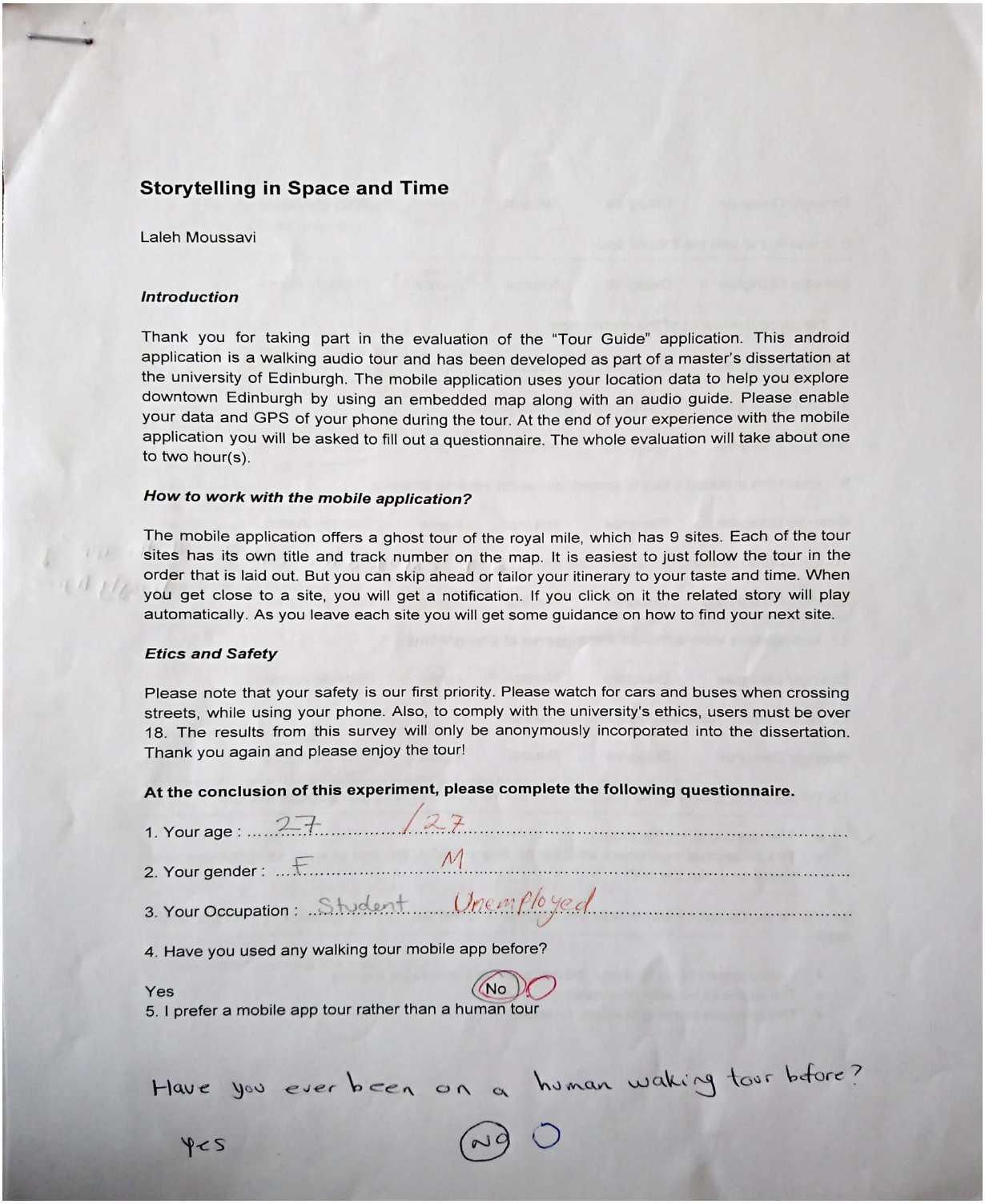
The canongate Tolbooth built in 1591, became a prison were many Covenanters were incarcerated. The scourge of those Covenanters, John Graham of Claverhouse, stayed here on his visits to Edinburgh. Which must have made the inmates even more uneasy. The building now houses the tolbooth tower which has a resident poltergeist who moves things around in the Bar area. **(Pre-Story)**
This one is not strictly a haunting, but worth including. Charles Dickens visited Canongate Kirkyard and saw a headstone which he thought read Ebenezer Scroggie: Mean man. In fact, the stone belonged to a corn dealer and said Ebenezer Scroggie: Meal man. Blissfully unaware of this, Dickens wrote A Christmas Carol, giving us the superb villain Ebenezer Scrooge and a classic supernatural trio in the ghosts of Christmas Past, Present and Future. The Tolbooth pub has poltergeist that moves things around and Chessel's Court is haunted by a woman wearing a black silk veil, who hanged herself in the 19th century. Queensberry House boasts a phantom kitchen boy, roasted and eaten by the mad son of the Marquess of Queensberry. Apparently he tasted of chicken.The last on may have been made up a bit. **(Story)**
Carry on to the end of the canongate and you will reach Abbey Strand and Holyrood. Behind it Salisbury Crags and Arthur's Seat, Edinburgh highest point, with magnificent vista over the city. **(Post story)**

Holyrood:

In the shadow of Arthur's Seat (where seventeen tiny coffins were discovered in 1836) is Holyrood. In 1128, King David I spotted a ghostly cross (or rood) between the antlers of a stag he was hunting, inspiring him to found Holyrood Abbey. Next came the Palace which, naturally, has an excellent class of ghost. It is supposedly haunted by Mary Queen of Scots and her husband Lord Darnley — which is a shame as they couldn't stand each other. The ghost of Mary's secretary, David Rizzio, who was stabbed to death by Darnley in front of the queen, also pops up. Some palace guides maintain the stain from his blood can still be seen on the floorboards, which is no mean feat, as they've been replaced more than once. Lowering the tone is the naked ghost of Bald Agnes, executed for witchcraft in 1592. **(Story)**

That is the end of the tour. it's only a glimpse into the city's Vicissitudes past. I hope you enjoy your time in this fascinating city. **(Post story)**

12 Appendix II: User Test Raw Data



Storytelling in Space and Time

Laleh Moussavi

Introduction

Thank you for taking part in the evaluation of the "Tour Guide" application. This android application is a walking audio tour and has been developed as part of a master's dissertation at the university of Edinburgh. The mobile application uses your location data to help you explore downtown Edinburgh by using an embedded map along with an audio guide. Please enable your data and GPS of your phone during the tour. At the end of your experience with the mobile application you will be asked to fill out a questionnaire. The whole evaluation will take about one to two hour(s).

How to work with the mobile application?

The mobile application offers a ghost tour of the royal mile, which has 9 sites. Each of the tour sites has its own title and track number on the map. It is easiest to just follow the tour in the order that is laid out. But you can skip ahead or tailor your itinerary to your taste and time. When you get close to a site, you will get a notification. If you click on it the related story will play automatically. As you leave each site you will get some guidance on how to find your next site.

Etics and Safety

Please note that your safety is our first priority. Please watch for cars and buses when crossing streets, while using your phone. Also, to comply with the university's ethics, users must be over 18. The results from this survey will only be anonymously incorporated into the dissertation. Thank you again and please enjoy the tour!

At the conclusion of this experiment, please complete the following questionnaire.

1. Your age : 27 / 27

2. Your gender : F M

3. Your Occupation : Student Unemployed

4. Have you used any walking tour mobile app before?

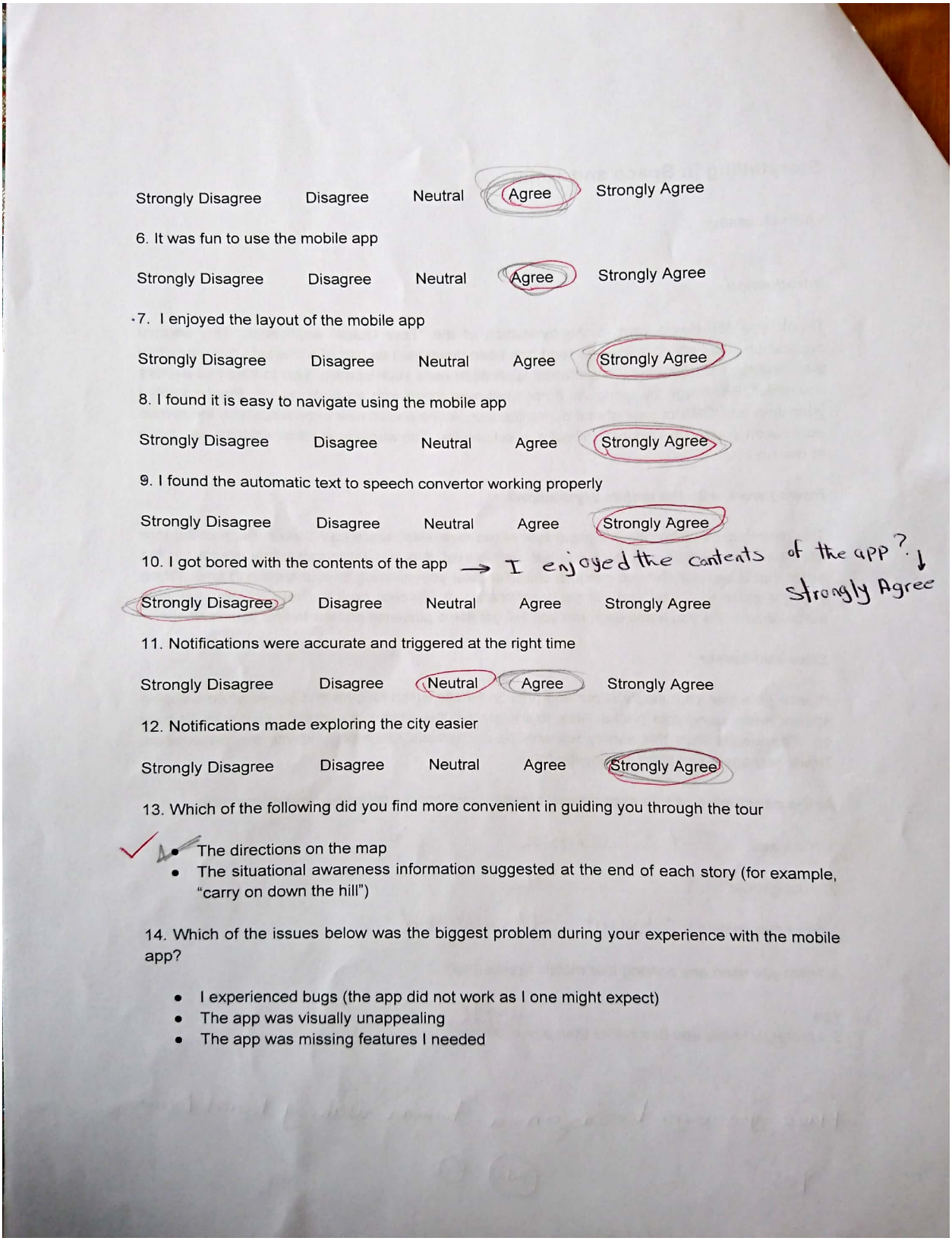
Yes No

5. I prefer a mobile app tour rather than a human tour

Have you ever been on a human waking tour before?

Yes NO

Figure 25. User 1 and 2, page 1.



Strongly Disagree Disagree Neutral Agree Strongly Agree

6. It was fun to use the mobile app

Strongly Disagree Disagree Neutral Agree Strongly Agree

7. I enjoyed the layout of the mobile app

Strongly Disagree Disagree Neutral Agree Strongly Agree

8. I found it is easy to navigate using the mobile app

Strongly Disagree Disagree Neutral Agree Strongly Agree

9. I found the automatic text to speech convertor working properly

Strongly Disagree Disagree Neutral Agree Strongly Agree

10. I got bored with the contents of the app → I enjoyed the contents of the app? ↓ Strongly Agree

Strongly Disagree Disagree Neutral Agree Strongly Agree

11. Notifications were accurate and triggered at the right time

Strongly Disagree Disagree Neutral Agree Strongly Agree

12. Notifications made exploring the city easier

Strongly Disagree Disagree Neutral Agree Strongly Agree

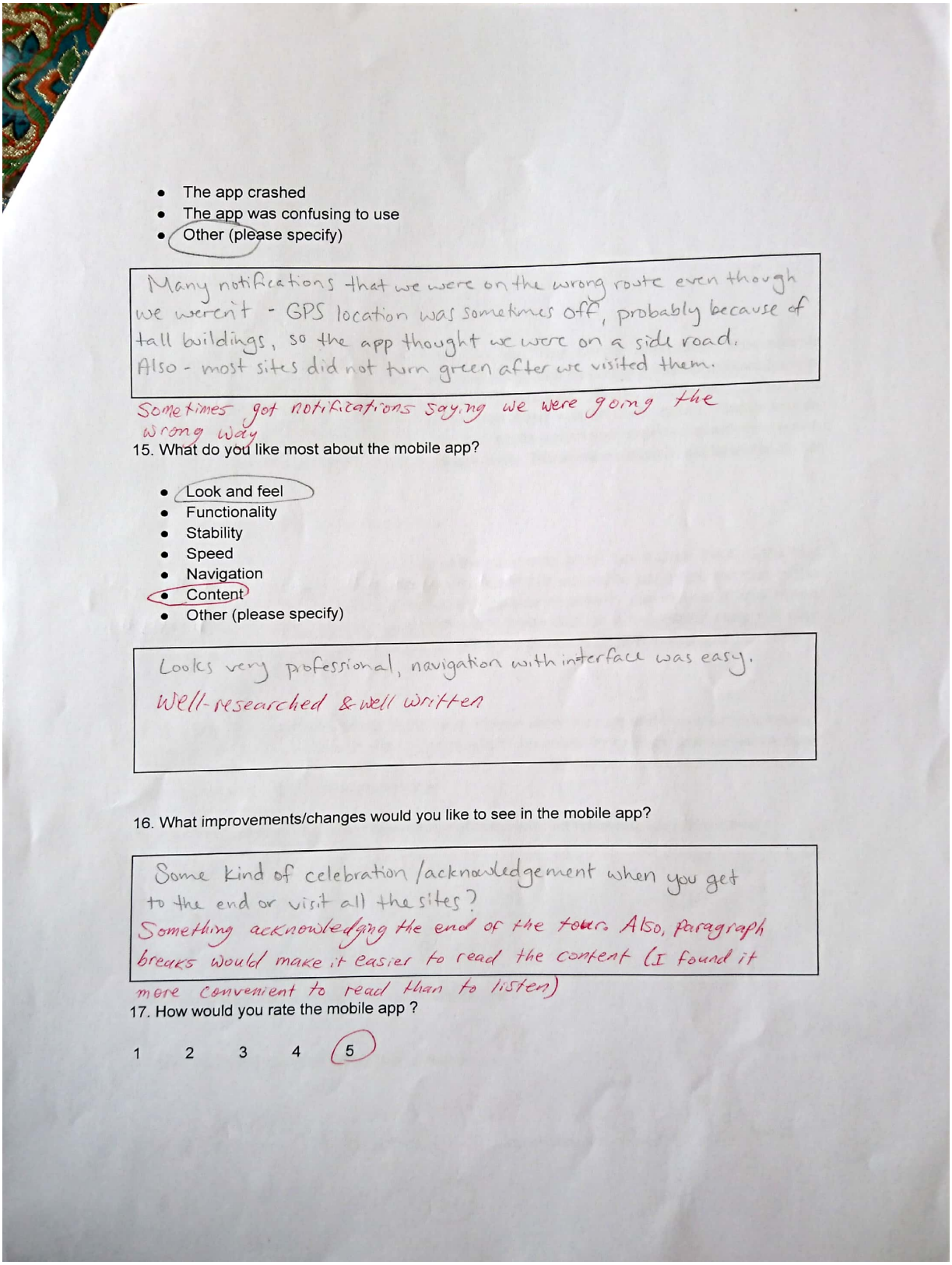
13. Which of the following did you find more convenient in guiding you through the tour

- The directions on the map
- The situational awareness information suggested at the end of each story (for example, "carry on down the hill")

14. Which of the issues below was the biggest problem during your experience with the mobile app?

- I experienced bugs (the app did not work as I one might expect)
- The app was visually unappealing
- The app was missing features I needed

Figure 26. User 1 and 2, page 2



- The app crashed
- The app was confusing to use
- Other (please specify)

Many notifications that we were on the wrong route even though we weren't - GPS location was sometimes off, probably because of tall buildings, so the app thought we were on a side road. Also - most sites did not turn green after we visited them.

Sometimes got notifications saying we were going the wrong way

15. What do you like most about the mobile app?

- Look and feel
- Functionality
- Stability
- Speed
- Navigation
- Content
- Other (please specify)

Looks very professional, navigation with interface was easy.

Well-researched & well written

16. What improvements/changes would you like to see in the mobile app?

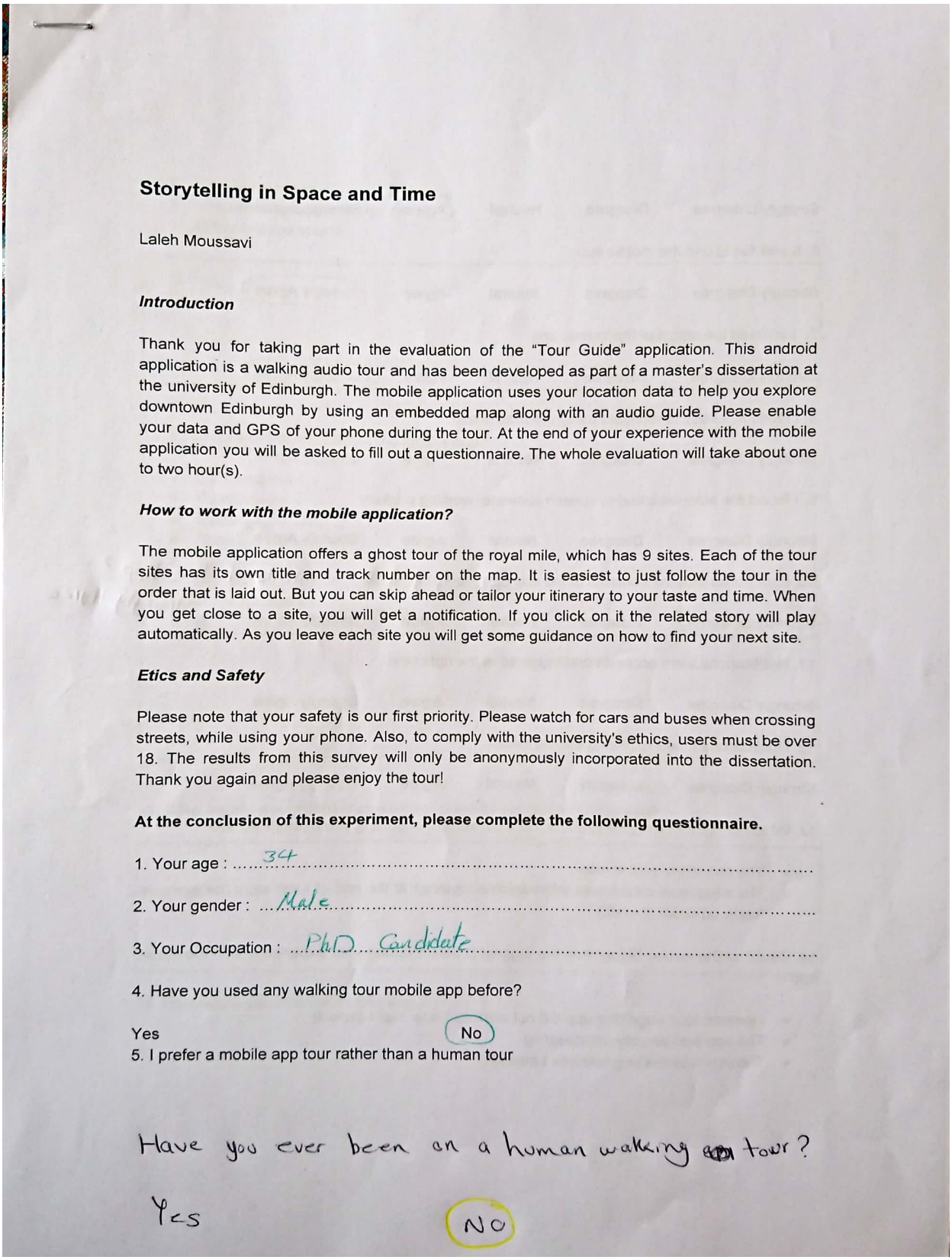
Some kind of celebration/acknowledgement when you get to the end or visit all the sites?

Something acknowledging the end of the tour. Also, paragraph breaks would make it easier to read the content (I found it more convenient to read than to listen)

17. How would you rate the mobile app ?

1 2 3 4 5

Figure 27. User 1 and 2, page 3



Storytelling in Space and Time

Laleh Moussavi

Introduction

Thank you for taking part in the evaluation of the "Tour Guide" application. This android application is a walking audio tour and has been developed as part of a master's dissertation at the university of Edinburgh. The mobile application uses your location data to help you explore downtown Edinburgh by using an embedded map along with an audio guide. Please enable your data and GPS of your phone during the tour. At the end of your experience with the mobile application you will be asked to fill out a questionnaire. The whole evaluation will take about one to two hour(s).

How to work with the mobile application?

The mobile application offers a ghost tour of the royal mile, which has 9 sites. Each of the tour sites has its own title and track number on the map. It is easiest to just follow the tour in the order that is laid out. But you can skip ahead or tailor your itinerary to your taste and time. When you get close to a site, you will get a notification. If you click on it the related story will play automatically. As you leave each site you will get some guidance on how to find your next site.

Etics and Safety

Please note that your safety is our first priority. Please watch for cars and buses when crossing streets, while using your phone. Also, to comply with the university's ethics, users must be over 18. The results from this survey will only be anonymously incorporated into the dissertation. Thank you again and please enjoy the tour!

At the conclusion of this experiment, please complete the following questionnaire.

1. Your age : 34

2. Your gender : Male

3. Your Occupation : PhD Candidate

4. Have you used any walking tour mobile app before?

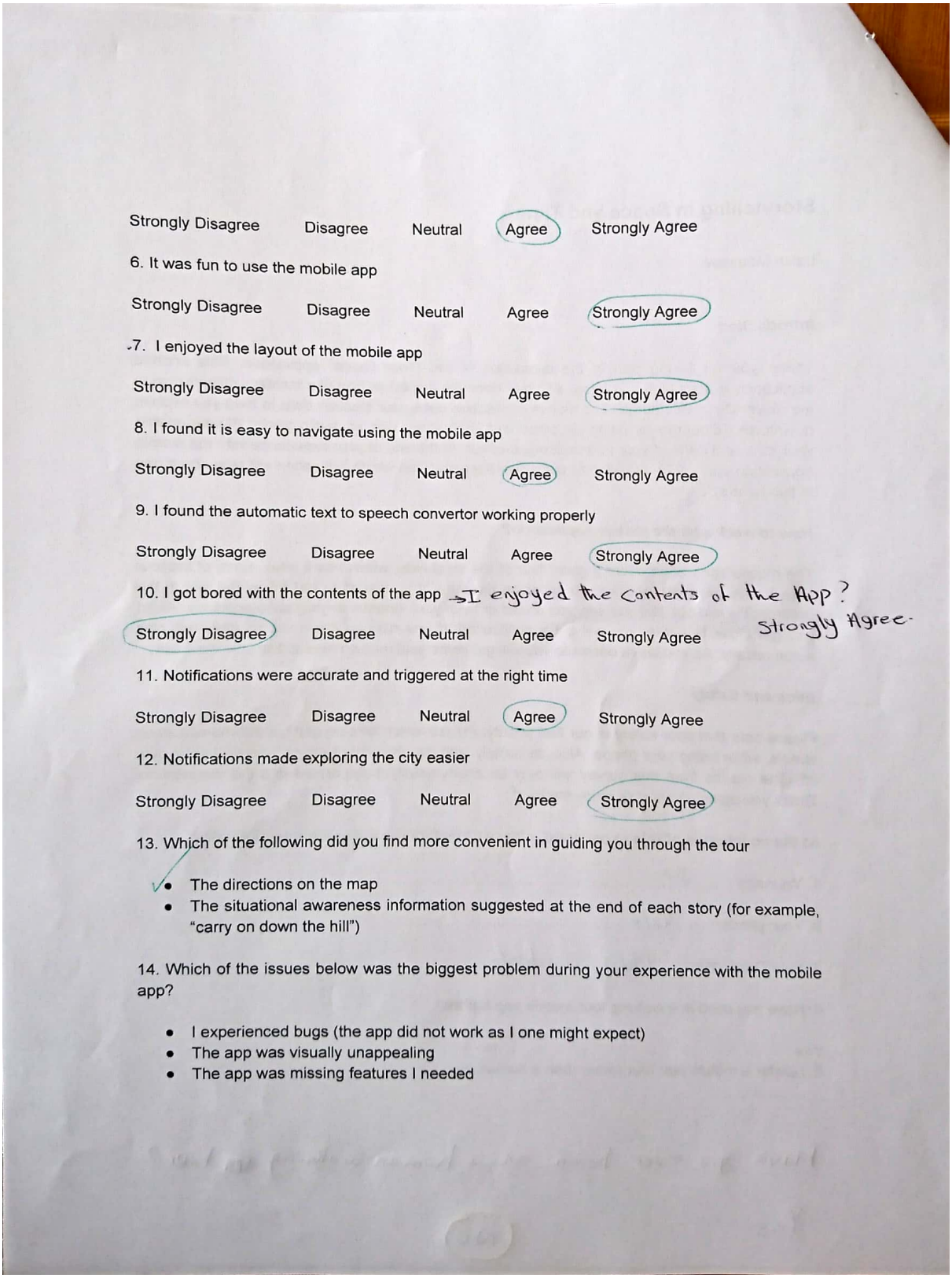
Yes No

5. I prefer a mobile app tour rather than a human tour

Have you ever been on a human walking tour?

Yes NO

Figure 28. User 3, page 1.



Strongly Disagree Disagree Neutral (Agree) Strongly Agree

6. It was fun to use the mobile app

Strongly Disagree Disagree Neutral Agree (Strongly Agree)

7. I enjoyed the layout of the mobile app

Strongly Disagree Disagree Neutral Agree (Strongly Agree)

8. I found it is easy to navigate using the mobile app

Strongly Disagree Disagree Neutral (Agree) Strongly Agree

9. I found the automatic text to speech convertor working properly

Strongly Disagree Disagree Neutral Agree (Strongly Agree)

10. I got bored with the contents of the app → I enjoyed the contents of the App? Strongly Agree.

(Strongly Disagree) Disagree Neutral Agree Strongly Agree

11. Notifications were accurate and triggered at the right time

Strongly Disagree Disagree Neutral (Agree) Strongly Agree

12. Notifications made exploring the city easier

Strongly Disagree Disagree Neutral Agree (Strongly Agree)

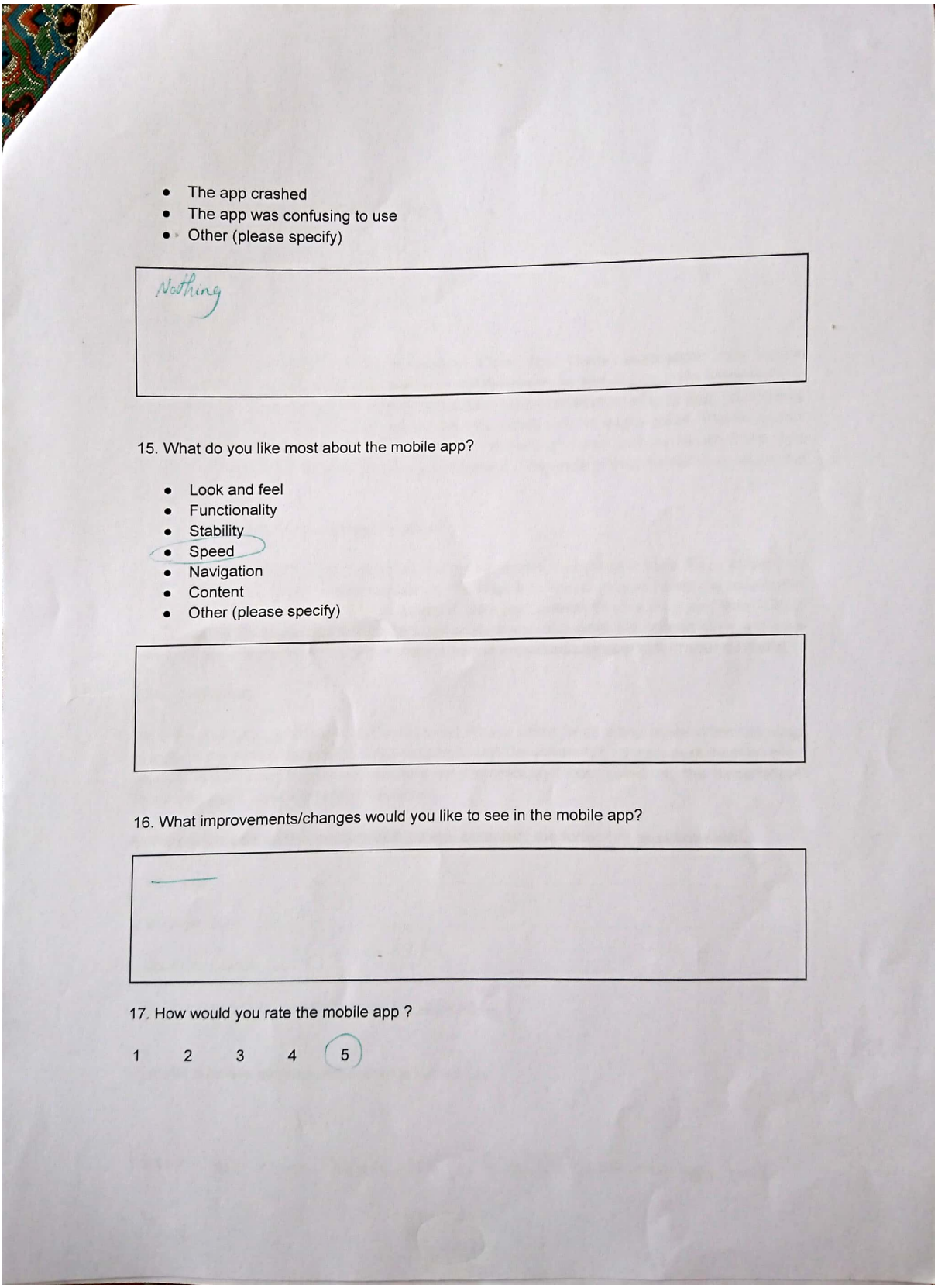
13. Which of the following did you find more convenient in guiding you through the tour

- ✓ The directions on the map
- The situational awareness information suggested at the end of each story (for example, "carry on down the hill")

14. Which of the issues below was the biggest problem during your experience with the mobile app?

- I experienced bugs (the app did not work as I one might expect)
- The app was visually unappealing
- The app was missing features I needed

Figure 29. User 3, page2.



- The app crashed
- The app was confusing to use
- Other (please specify)

Nothing

15. What do you like most about the mobile app?

- Look and feel
- Functionality
- Stability
- Speed
- Navigation
- Content
- Other (please specify)

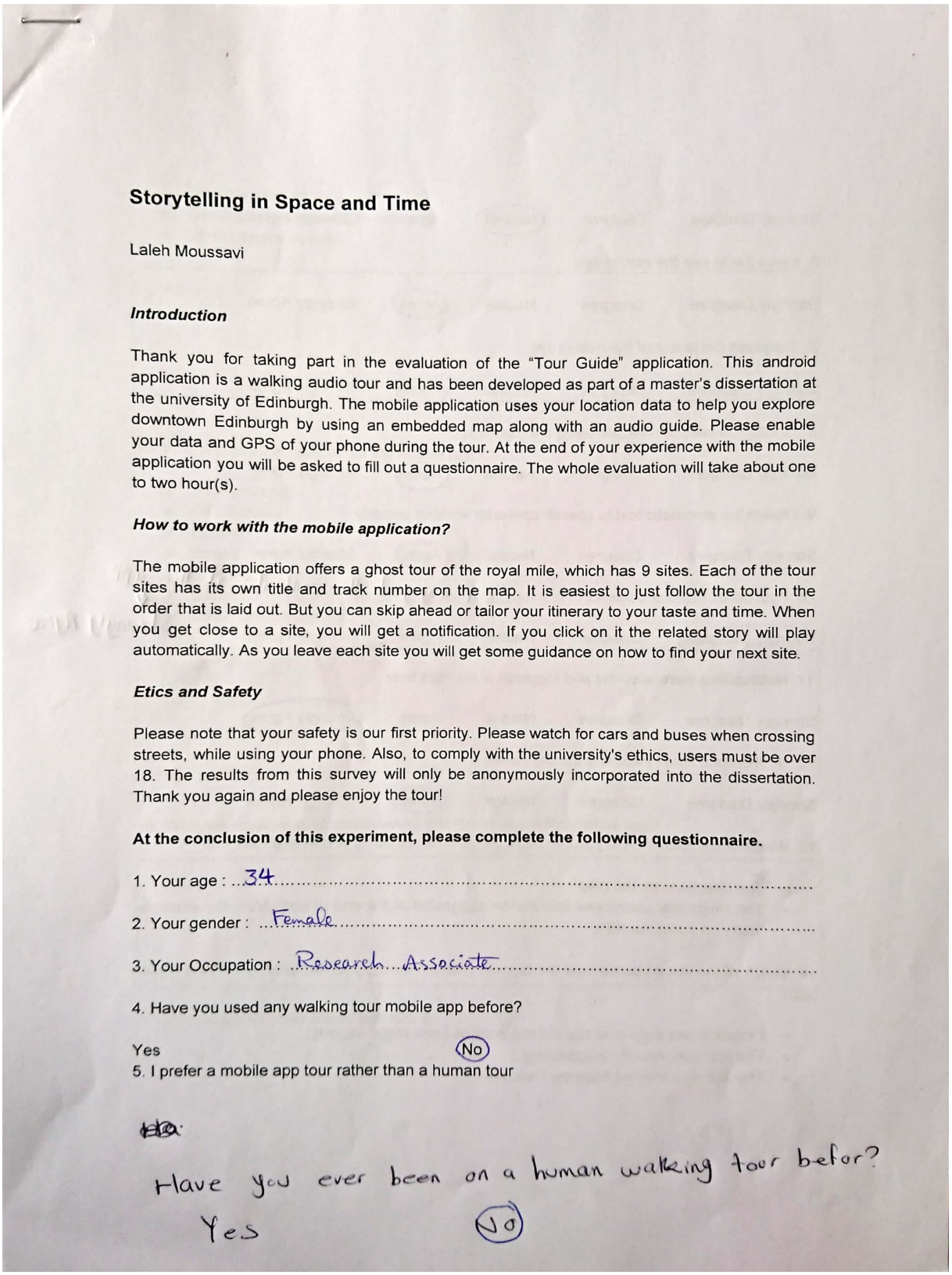
16. What improvements/changes would you like to see in the mobile app?

—

17. How would you rate the mobile app ?

1 2 3 4 5

Figure 30. User 3, page 3.



Storytelling in Space and Time

Laleh Moussavi

Introduction

Thank you for taking part in the evaluation of the "Tour Guide" application. This android application is a walking audio tour and has been developed as part of a master's dissertation at the university of Edinburgh. The mobile application uses your location data to help you explore downtown Edinburgh by using an embedded map along with an audio guide. Please enable your data and GPS of your phone during the tour. At the end of your experience with the mobile application you will be asked to fill out a questionnaire. The whole evaluation will take about one to two hour(s).

How to work with the mobile application?

The mobile application offers a ghost tour of the royal mile, which has 9 sites. Each of the tour sites has its own title and track number on the map. It is easiest to just follow the tour in the order that is laid out. But you can skip ahead or tailor your itinerary to your taste and time. When you get close to a site, you will get a notification. If you click on it the related story will play automatically. As you leave each site you will get some guidance on how to find your next site.

Etics and Safety

Please note that your safety is our first priority. Please watch for cars and buses when crossing streets, while using your phone. Also, to comply with the university's ethics, users must be over 18. The results from this survey will only be anonymously incorporated into the dissertation. Thank you again and please enjoy the tour!

At the conclusion of this experiment, please complete the following questionnaire.

1. Your age : 34

2. Your gender : Female

3. Your Occupation : Research Associate

4. Have you used any walking tour mobile app before?

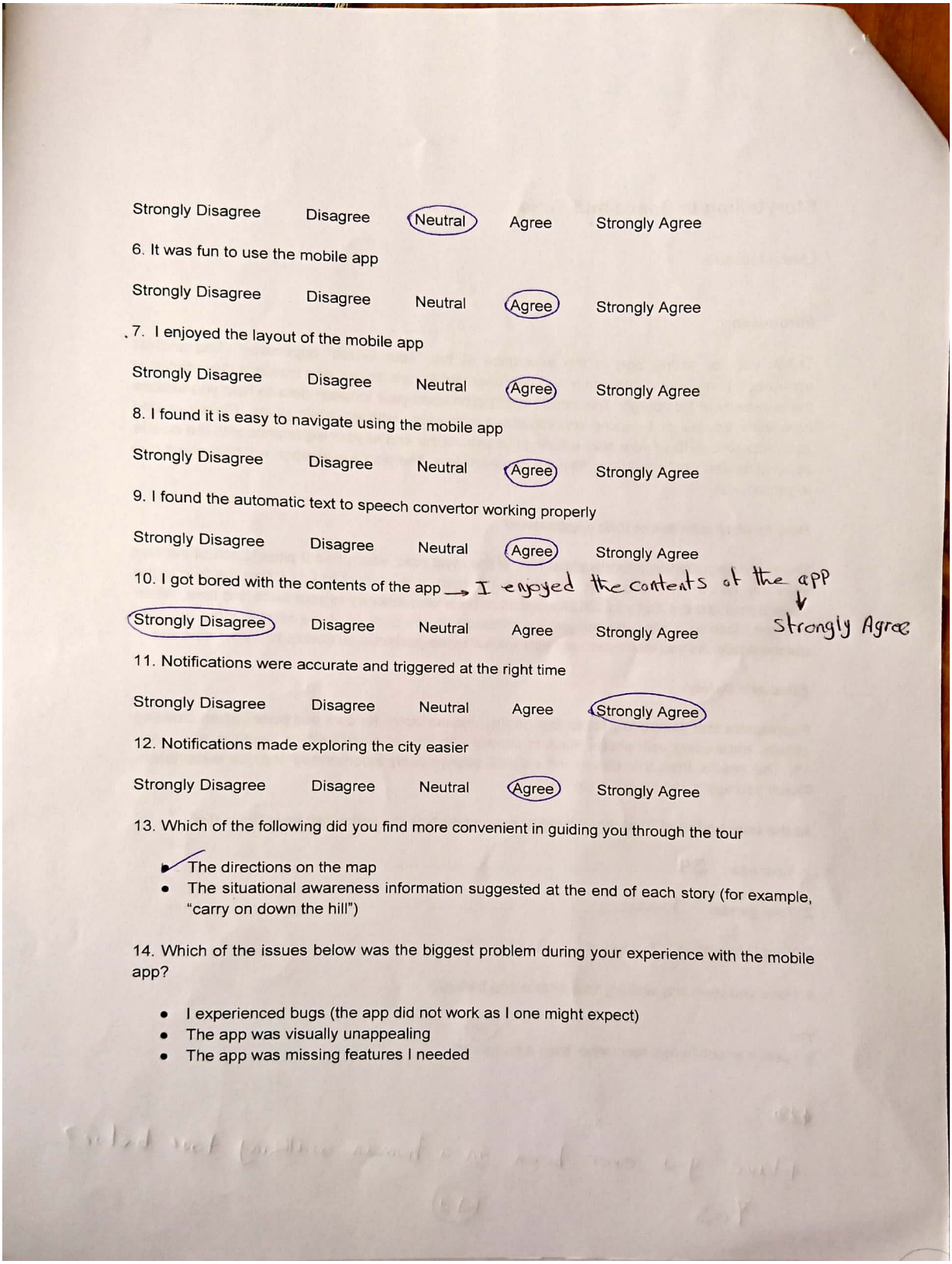
Yes (No)

5. I prefer a mobile app tour rather than a human tour

Have you ever been on a human walking tour befor?

Yes (No)

Figure 31. User 4, page 1.



Strongly Disagree Disagree (Neutral) Agree Strongly Agree

6. It was fun to use the mobile app

Strongly Disagree Disagree Neutral (Agree) Strongly Agree

7. I enjoyed the layout of the mobile app

Strongly Disagree Disagree Neutral (Agree) Strongly Agree

8. I found it is easy to navigate using the mobile app

Strongly Disagree Disagree Neutral (Agree) Strongly Agree

9. I found the automatic text to speech convertor working properly

Strongly Disagree Disagree Neutral (Agree) Strongly Agree

10. I got bored with the contents of the app → I enjoyed the contents of the app ↓ Strongly Agree

(Strongly Disagree) Disagree Neutral Agree Strongly Agree

11. Notifications were accurate and triggered at the right time

Strongly Disagree Disagree Neutral Agree (Strongly Agree)

12. Notifications made exploring the city easier

Strongly Disagree Disagree Neutral (Agree) Strongly Agree

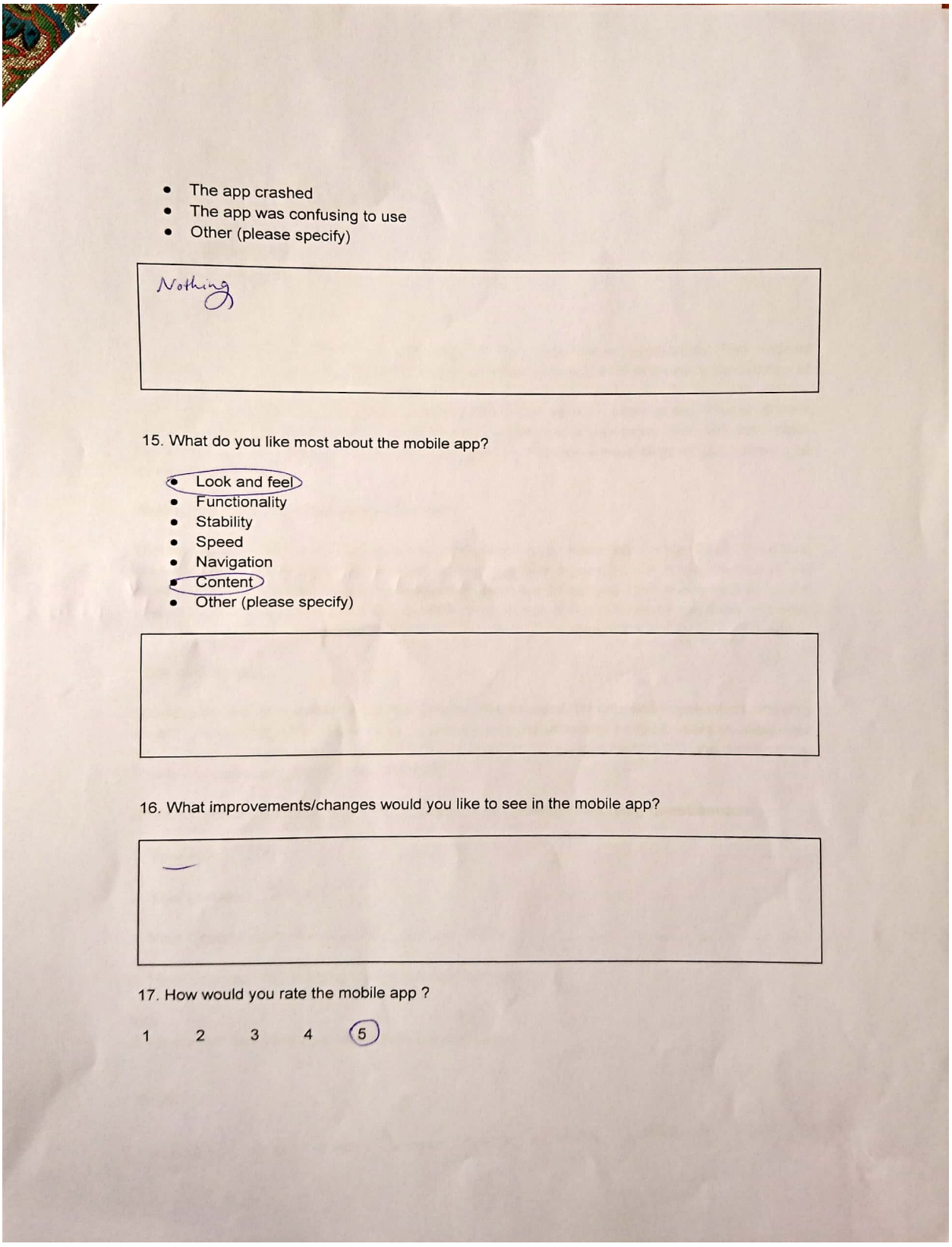
13. Which of the following did you find more convenient in guiding you through the tour

- ✓ The directions on the map
- The situational awareness information suggested at the end of each story (for example, "carry on down the hill")

14. Which of the issues below was the biggest problem during your experience with the mobile app?

- I experienced bugs (the app did not work as I one might expect)
- The app was visually unappealing
- The app was missing features I needed

Figure 32. User 4, page 2.



- The app crashed
- The app was confusing to use
- Other (please specify)

Nothing

15. What do you like most about the mobile app?

- Look and feel
- Functionality
- Stability
- Speed
- Navigation
- Content
- Other (please specify)

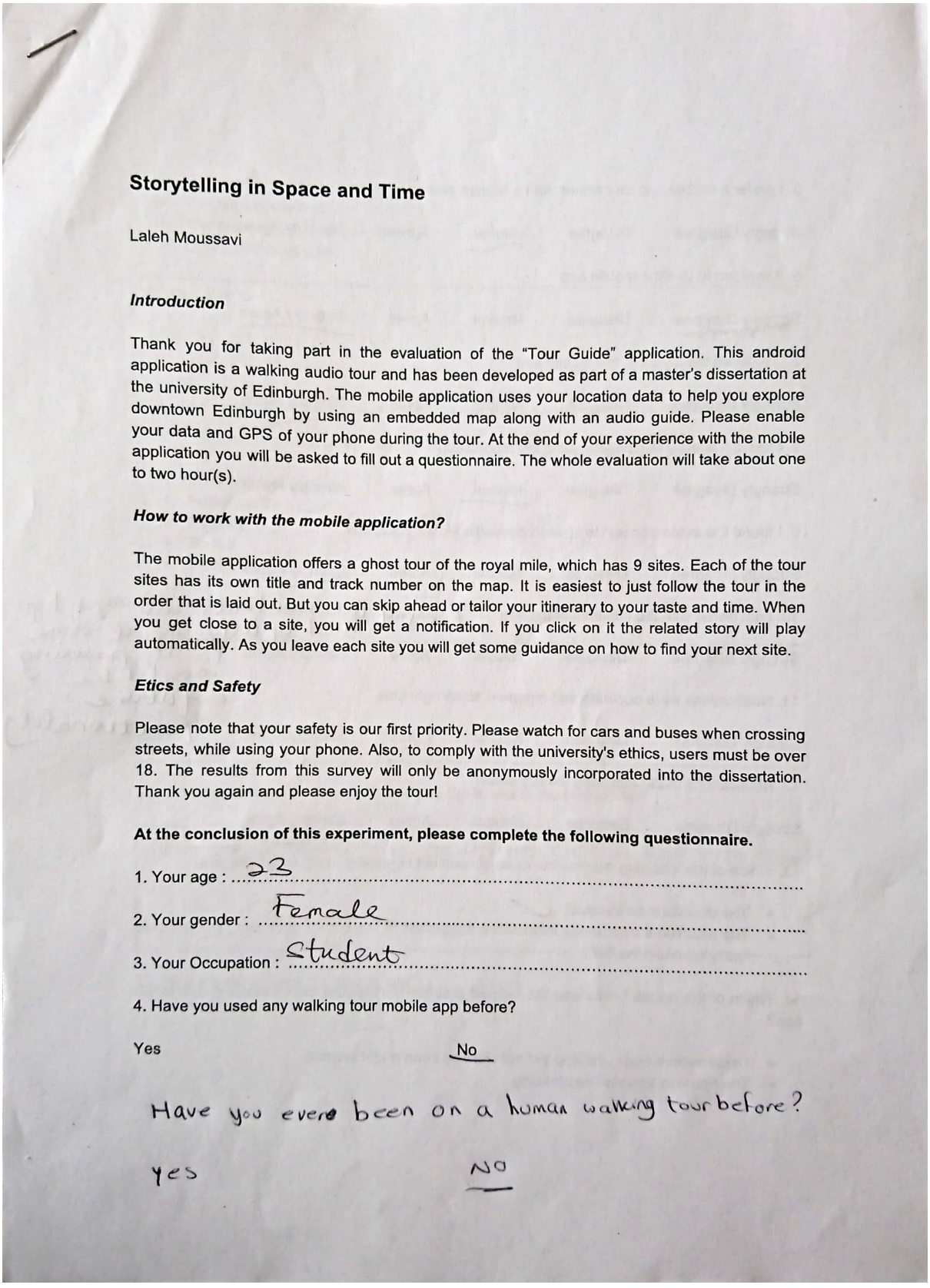
16. What improvements/changes would you like to see in the mobile app?

—

17. How would you rate the mobile app ?

1 2 3 4 5

Figure 33. User 4, page 3.



Storytelling in Space and Time

Laleh Moussavi

Introduction

Thank you for taking part in the evaluation of the "Tour Guide" application. This android application is a walking audio tour and has been developed as part of a master's dissertation at the university of Edinburgh. The mobile application uses your location data to help you explore downtown Edinburgh by using an embedded map along with an audio guide. Please enable your data and GPS of your phone during the tour. At the end of your experience with the mobile application you will be asked to fill out a questionnaire. The whole evaluation will take about one to two hour(s).

How to work with the mobile application?

The mobile application offers a ghost tour of the royal mile, which has 9 sites. Each of the tour sites has its own title and track number on the map. It is easiest to just follow the tour in the order that is laid out. But you can skip ahead or tailor your itinerary to your taste and time. When you get close to a site, you will get a notification. If you click on it the related story will play automatically. As you leave each site you will get some guidance on how to find your next site.

Etics and Safety

Please note that your safety is our first priority. Please watch for cars and buses when crossing streets, while using your phone. Also, to comply with the university's ethics, users must be over 18. The results from this survey will only be anonymously incorporated into the dissertation. Thank you again and please enjoy the tour!

At the conclusion of this experiment, please complete the following questionnaire.

1. Your age : 23

2. Your gender : Female

3. Your Occupation : Student

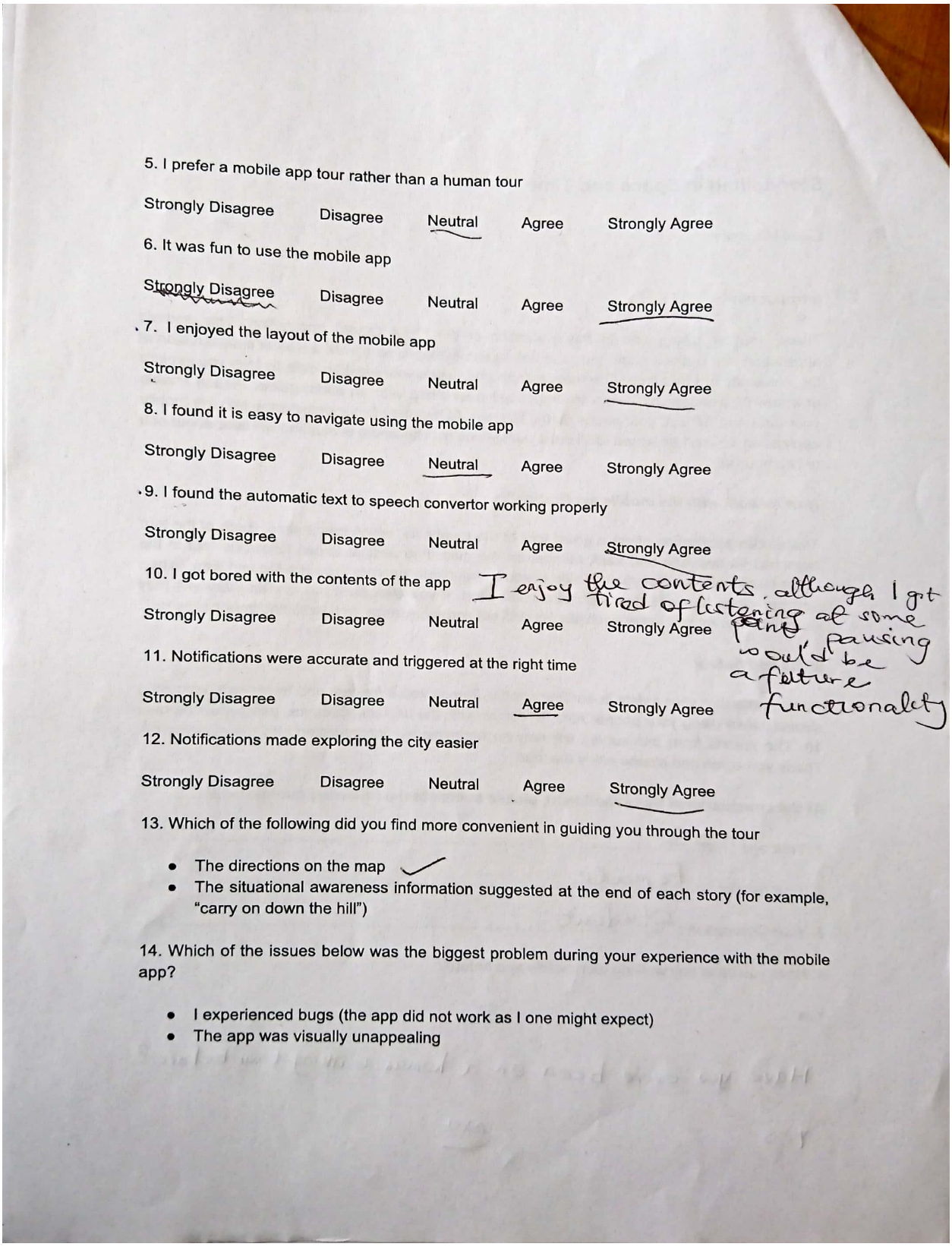
4. Have you used any walking tour mobile app before?

Yes No

Have you ever been on a human walking tour before?

yes NO

Figure 34. User 5, page 1.



5. I prefer a mobile app tour rather than a human tour

Strongly Disagree Disagree Neutral Agree Strongly Agree

6. It was fun to use the mobile app

Strongly Disagree Disagree Neutral Agree Strongly Agree

7. I enjoyed the layout of the mobile app

Strongly Disagree Disagree Neutral Agree Strongly Agree

8. I found it is easy to navigate using the mobile app

Strongly Disagree Disagree Neutral Agree Strongly Agree

9. I found the automatic text to speech convertor working properly

Strongly Disagree Disagree Neutral Agree Strongly Agree

10. I got bored with the contents of the app

I enjoy the contents although I got tired of listening at some point pausing would be a feature functionality

Strongly Disagree Disagree Neutral Agree Strongly Agree

11. Notifications were accurate and triggered at the right time

Strongly Disagree Disagree Neutral Agree Strongly Agree

12. Notifications made exploring the city easier

Strongly Disagree Disagree Neutral Agree Strongly Agree

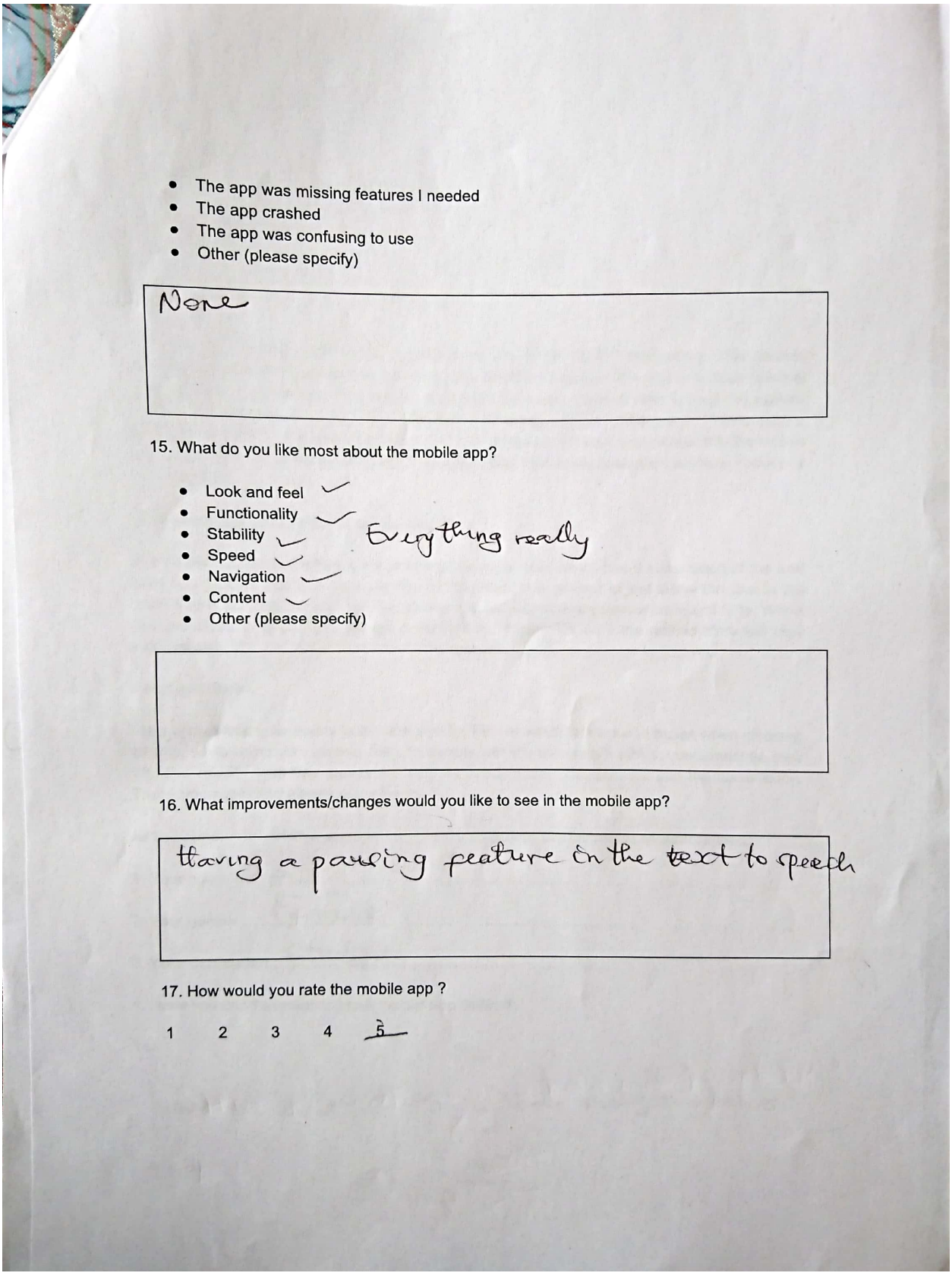
13. Which of the following did you find more convenient in guiding you through the tour

- The directions on the map
- The situational awareness information suggested at the end of each story (for example, "carry on down the hill")

14. Which of the issues below was the biggest problem during your experience with the mobile app?

- I experienced bugs (the app did not work as I one might expect)
- The app was visually unappealing

Figure 35. User 5, page 2..



- The app was missing features I needed
- The app crashed
- The app was confusing to use
- Other (please specify)

None

15. What do you like most about the mobile app?

- Look and feel ✓
- Functionality ✓
- Stability ✓ Everything really
- Speed ✓
- Navigation ✓
- Content ✓
- Other (please specify)

16. What improvements/changes would you like to see in the mobile app?

having a pausing feature in the ~~text~~ to speech

17. How would you rate the mobile app ?

1 2 3 4 5

Figure 36. User 5, page 3.

Storytelling in Space and Time

Laleh Moussavi

Introduction

Thank you for taking part in the evaluation of the "Tour Guide" application. This android application is a walking audio tour and has been developed as part of a master's dissertation at the university of Edinburgh. The mobile application uses your location data to help you explore downtown Edinburgh by using an embedded map along with an audio guide. Please enable your data and GPS of your phone during the tour. At the end of your experience with the mobile application you will be asked to fill out a questionnaire. The whole evaluation will take about one to two hour(s).

How to work with the mobile application?

The mobile application offers a ghost tour of the royal mile, which has 9 sites. Each of the tour sites has its own title and track number on the map. It is best to just follow the tour in the order that is laid out. But you can skip ahead or tailor your itinerary to your taste and time. When you get close to a site, you will get a notification. If you click on it the related story will play automatically. As you leave each site you will get some guidance on how to find your next site.

Etics and Safety

Please note that your safety is our first priority. Please watch for cars and buses when crossing streets, while using your phone. Also, to comply with the university's ethics, users must be over 18. The results from this survey will only be anonymously incorporated into the dissertation. Thank you again and please enjoy the tour!

At the conclusion of this experiment, please complete the following questionnaire.

1. Your age : 23

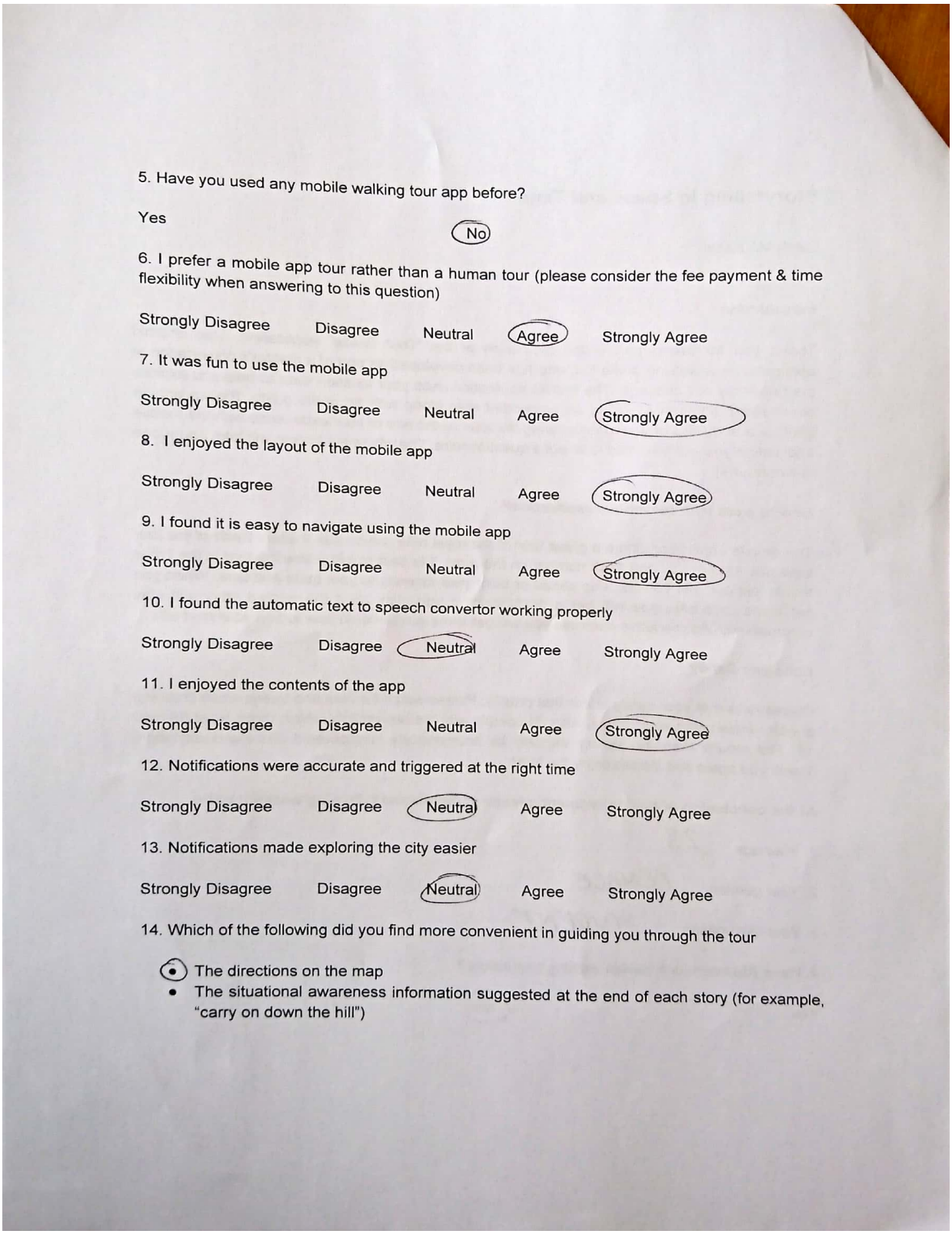
2. Your gender : FEMALE

3. Your Occupation : STUDENT

4. Have you been on a human walking tour before?

Yes (No)

Figure 37. User 6, page 1.



5. Have you used any mobile walking tour app before?

Yes (No)

6. I prefer a mobile app tour rather than a human tour (please consider the fee payment & time flexibility when answering to this question)

Strongly Disagree Disagree Neutral (Agree) Strongly Agree

7. It was fun to use the mobile app

Strongly Disagree Disagree Neutral Agree (Strongly Agree)

8. I enjoyed the layout of the mobile app

Strongly Disagree Disagree Neutral Agree (Strongly Agree)

9. I found it is easy to navigate using the mobile app

Strongly Disagree Disagree Neutral Agree (Strongly Agree)

10. I found the automatic text to speech convertor working properly

Strongly Disagree Disagree (Neutral) Agree Strongly Agree

11. I enjoyed the contents of the app

Strongly Disagree Disagree Neutral Agree (Strongly Agree)

12. Notifications were accurate and triggered at the right time

Strongly Disagree Disagree (Neutral) Agree Strongly Agree

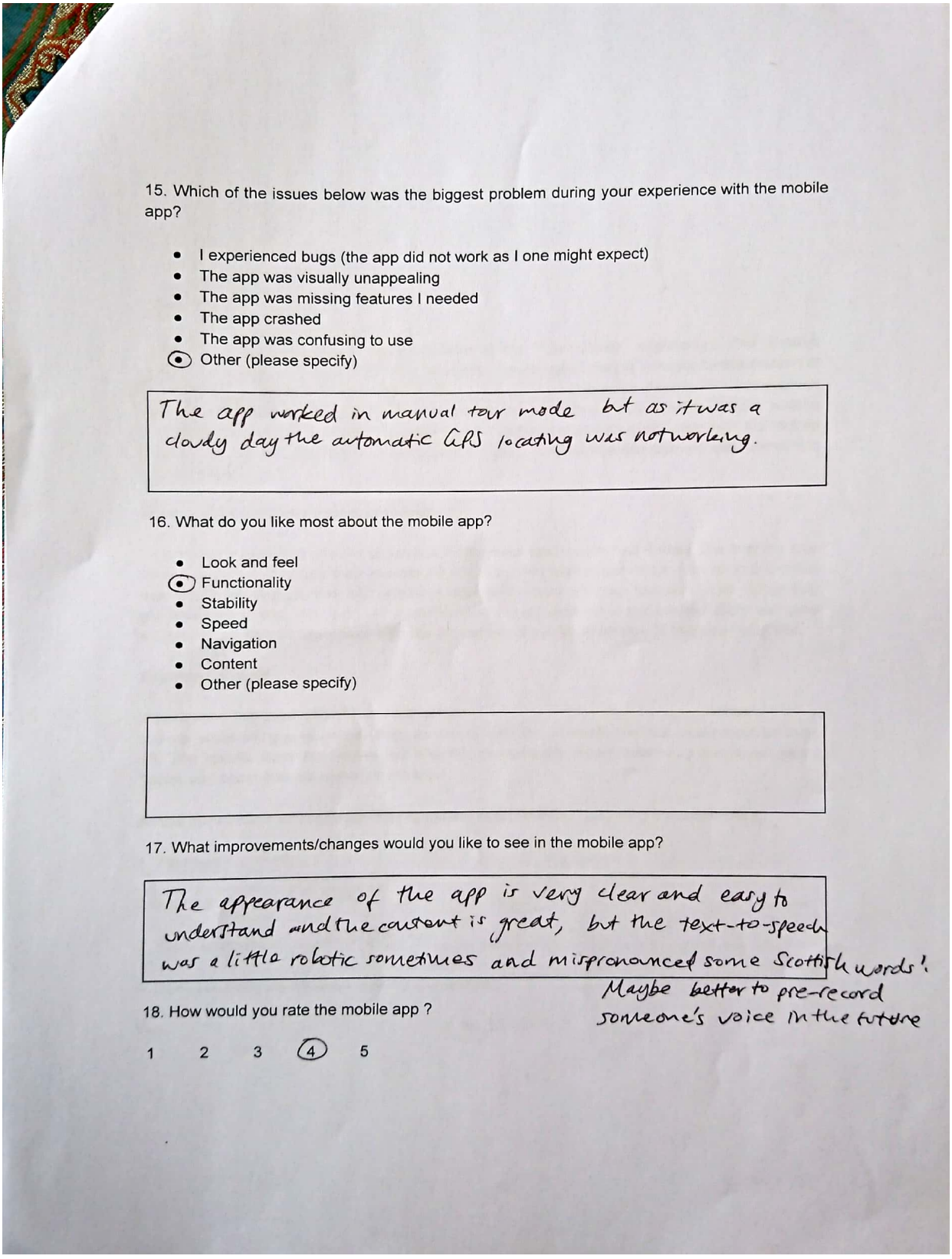
13. Notifications made exploring the city easier

Strongly Disagree Disagree (Neutral) Agree Strongly Agree

14. Which of the following did you find more convenient in guiding you through the tour

(•) The directions on the map
- The situational awareness information suggested at the end of each story (for example, "carry on down the hill")

Figure 38. User 6, page 2.



15. Which of the issues below was the biggest problem during your experience with the mobile app?

- I experienced bugs (the app did not work as I one might expect)
- The app was visually unappealing
- The app was missing features I needed
- The app crashed
- The app was confusing to use
- (•) Other (please specify)

The app worked in manual tour mode but as it was a cloudy day the automatic GPS locating was not working.

16. What do you like most about the mobile app?

- Look and feel
- (•) Functionality
- Stability
- Speed
- Navigation
- Content
- Other (please specify)

17. What improvements/changes would you like to see in the mobile app?

The appearance of the app is very clear and easy to understand and the content is great, but the text-to-speech was a little robotic sometimes and mispronounced some Scottish words! Maybe better to pre-record someone's voice in the future

18. How would you rate the mobile app ?

1 2 3 (4) 5

Figure 39. User 6, page 3.

Storytelling in Space and Time

Laleh Moussavi

Introduction

Thank you for taking part in the evaluation of the "Tour Guide" application. This android application is a walking audio tour and has been developed as part of a master's dissertation at the university of Edinburgh. The mobile application uses your location data to help you explore downtown Edinburgh by using an embedded map along with an audio guide. Please enable your data and GPS of your phone during the tour. At the end of your experience with the mobile application you will be asked to fill out a questionnaire. The whole evaluation will take about one to two hour(s).

How to work with the mobile application?

The mobile application offers a ghost tour of the royal mile, which has 9 sites. Each of the tour sites has its own title and track number on the map. It is best to just follow the tour in the order that is laid out. But you can skip ahead or tailor your itinerary to your taste and time. When you get close to a site, you will get a notification. If you click on it the related story will play automatically. As you leave each site you will get some guidance on how to find your next site.

Ethics?

Etics and Safety

Please note that your safety is our first priority. Please watch for cars and buses when crossing streets, while using your phone. Also, to comply with the university's ethics, users must be over 18. The results from this survey will only be anonymously incorporated into the dissertation. Thank you again and please enjoy the tour!

At the conclusion of this experiment, please complete the following questionnaire.

1. Your age : 28

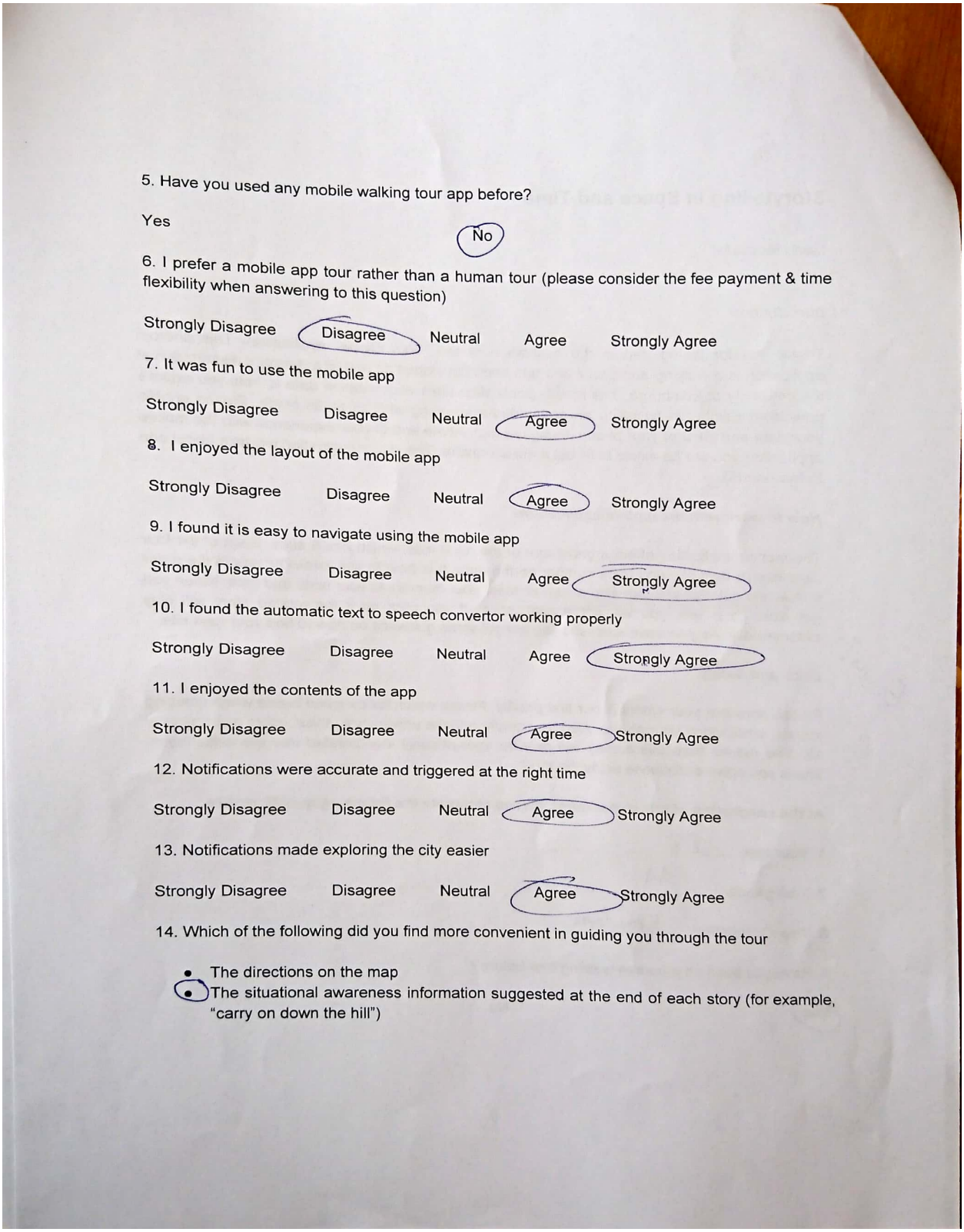
2. Your gender : M

3. Your Occupation : Student

4. Have you been on a human walking tour before?

(Yes) No

Figure 40. User 7, page 1.



5. Have you used any mobile walking tour app before?

Yes

No

6. I prefer a mobile app tour rather than a human tour (please consider the fee payment & time flexibility when answering to this question)

Strongly Disagree | Disagree | Neutral | Agree | Strongly Agree

7. It was fun to use the mobile app

Strongly Disagree | Disagree | Neutral | Agree | Strongly Agree

8. I enjoyed the layout of the mobile app

Strongly Disagree | Disagree | Neutral | Agree | Strongly Agree

9. I found it is easy to navigate using the mobile app

Strongly Disagree | Disagree | Neutral | Agree | Strongly Agree

10. I found the automatic text to speech convertor working properly

Strongly Disagree | Disagree | Neutral | Agree | Strongly Agree

11. I enjoyed the contents of the app

Strongly Disagree | Disagree | Neutral | Agree | Strongly Agree

12. Notifications were accurate and triggered at the right time

Strongly Disagree | Disagree | Neutral | Agree | Strongly Agree

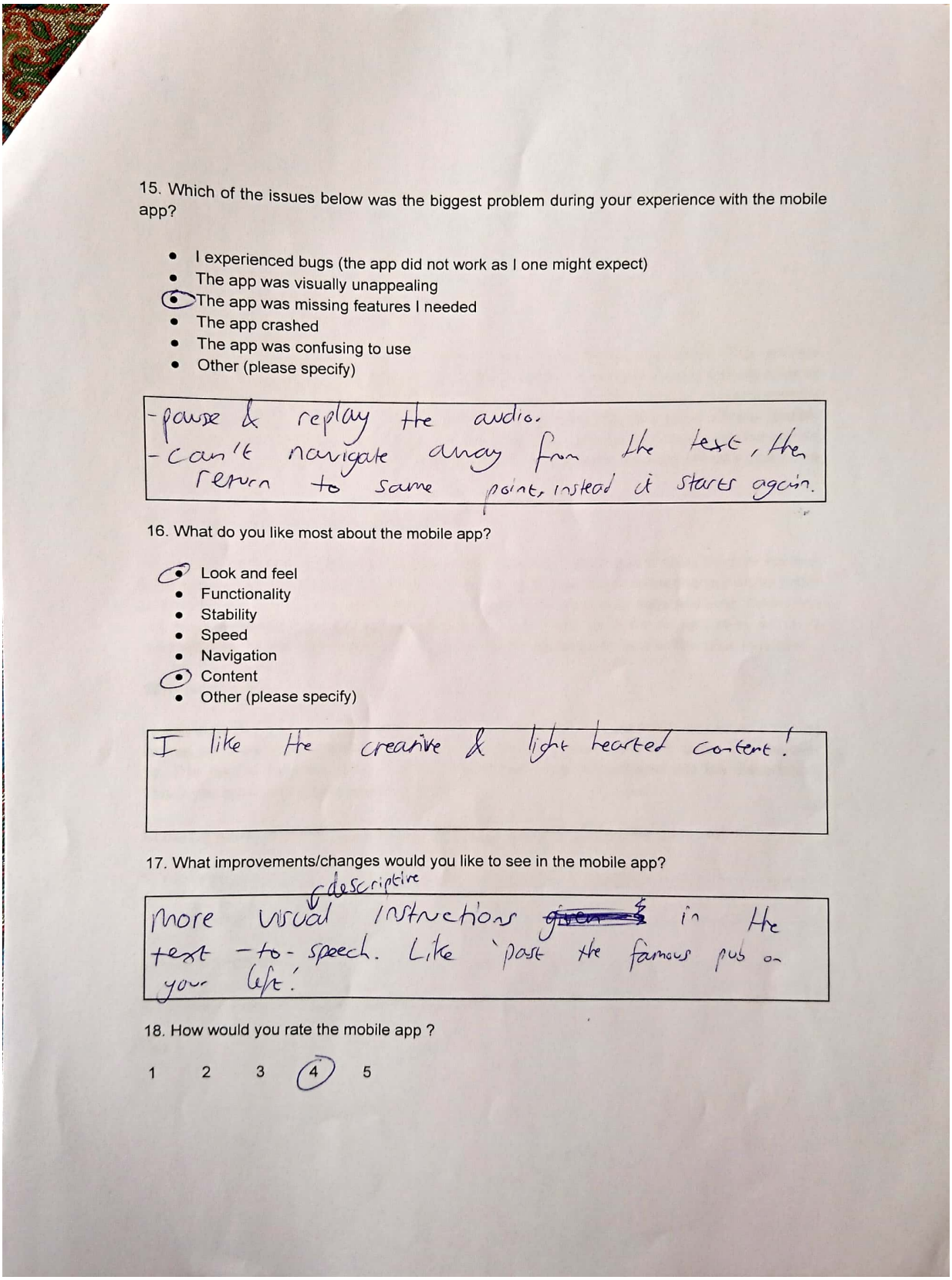
13. Notifications made exploring the city easier

Strongly Disagree | Disagree | Neutral | Agree | Strongly Agree

14. Which of the following did you find more convenient in guiding you through the tour

- The directions on the map
- The situational awareness information suggested at the end of each story (for example, "carry on down the hill")

Figure 41. User 7, page 2.



15. Which of the issues below was the biggest problem during your experience with the mobile app?

- I experienced bugs (the app did not work as I one might expect)
- The app was visually unappealing
- (•) The app was missing features I needed
- The app crashed
- The app was confusing to use
- Other (please specify)

- pause & replay the audio.
- can't navigate away from the text, then return to same point, instead it starts again.

16. What do you like most about the mobile app?

- (•) Look and feel
- Functionality
- Stability
- Speed
- Navigation
- (•) Content
- Other (please specify)

I like the creative & light hearted content!

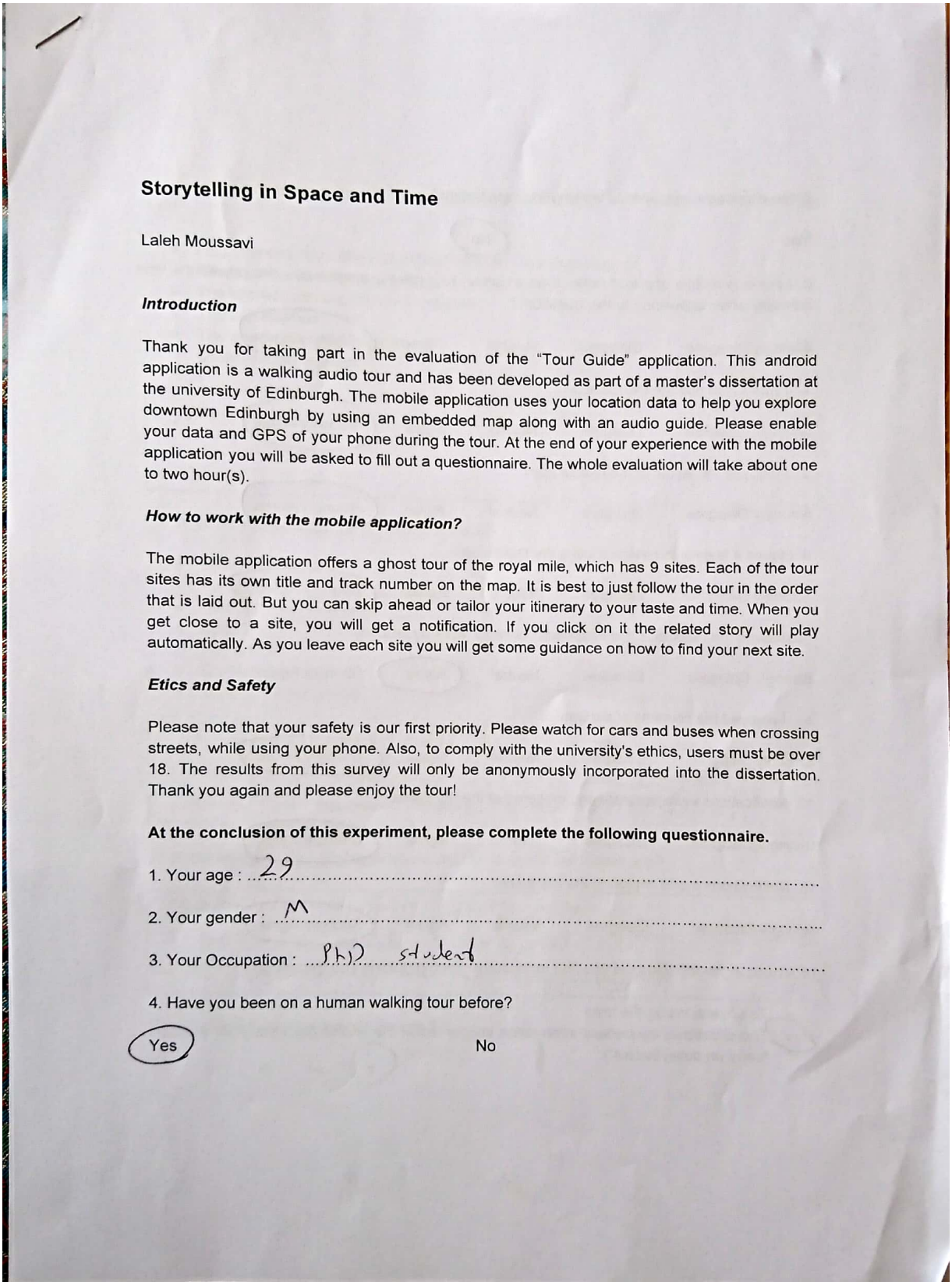
17. What improvements/changes would you like to see in the mobile app?

more descriptive visual instructions ~~given~~ in the text-to-speech. Like 'past the famous pub on your left'.

18. How would you rate the mobile app ?

1 2 3 (4) 5

Figure 42. User 7, page 3.



Storytelling in Space and Time

Laleh Moussavi

Introduction

Thank you for taking part in the evaluation of the "Tour Guide" application. This android application is a walking audio tour and has been developed as part of a master's dissertation at the university of Edinburgh. The mobile application uses your location data to help you explore downtown Edinburgh by using an embedded map along with an audio guide. Please enable your data and GPS of your phone during the tour. At the end of your experience with the mobile application you will be asked to fill out a questionnaire. The whole evaluation will take about one to two hour(s).

How to work with the mobile application?

The mobile application offers a ghost tour of the royal mile, which has 9 sites. Each of the tour sites has its own title and track number on the map. It is best to just follow the tour in the order that is laid out. But you can skip ahead or tailor your itinerary to your taste and time. When you get close to a site, you will get a notification. If you click on it the related story will play automatically. As you leave each site you will get some guidance on how to find your next site.

Etics and Safety

Please note that your safety is our first priority. Please watch for cars and buses when crossing streets, while using your phone. Also, to comply with the university's ethics, users must be over 18. The results from this survey will only be anonymously incorporated into the dissertation. Thank you again and please enjoy the tour!

At the conclusion of this experiment, please complete the following questionnaire.

1. Your age : 29

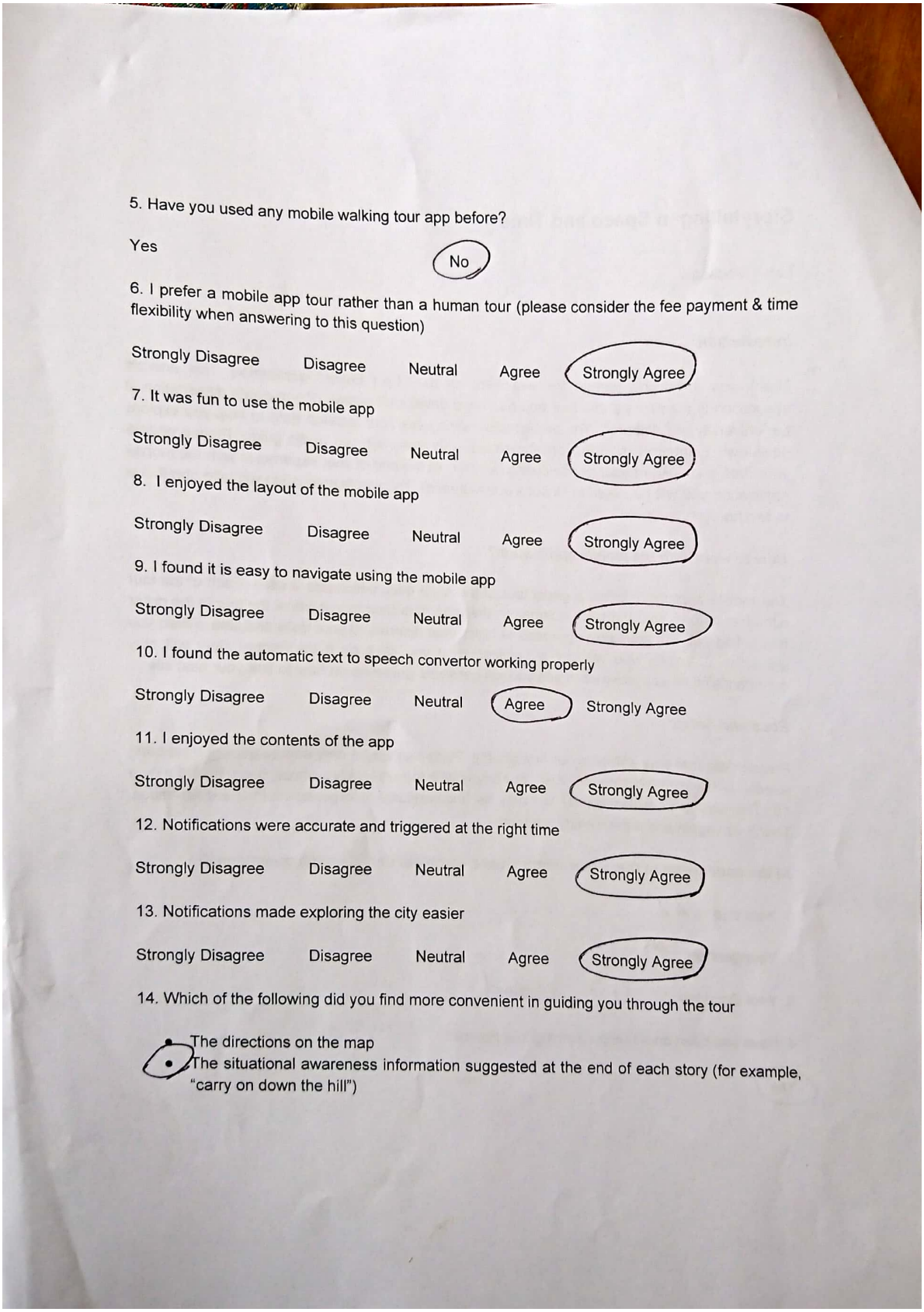
2. Your gender : M

3. Your Occupation : PhD student

4. Have you been on a human walking tour before?

Yes (circled) No

Figure 43. User 8, page 1



5. Have you used any mobile walking tour app before?

Yes (No)

6. I prefer a mobile app tour rather than a human tour (please consider the fee payment & time flexibility when answering to this question)

Strongly Disagree Disagree Neutral Agree (Strongly Agree)

7. It was fun to use the mobile app

Strongly Disagree Disagree Neutral Agree (Strongly Agree)

8. I enjoyed the layout of the mobile app

Strongly Disagree Disagree Neutral Agree (Strongly Agree)

9. I found it is easy to navigate using the mobile app

Strongly Disagree Disagree Neutral Agree (Strongly Agree)

10. I found the automatic text to speech convertor working properly

Strongly Disagree Disagree Neutral (Agree) Strongly Agree

11. I enjoyed the contents of the app

Strongly Disagree Disagree Neutral Agree (Strongly Agree)

12. Notifications were accurate and triggered at the right time

Strongly Disagree Disagree Neutral Agree (Strongly Agree)

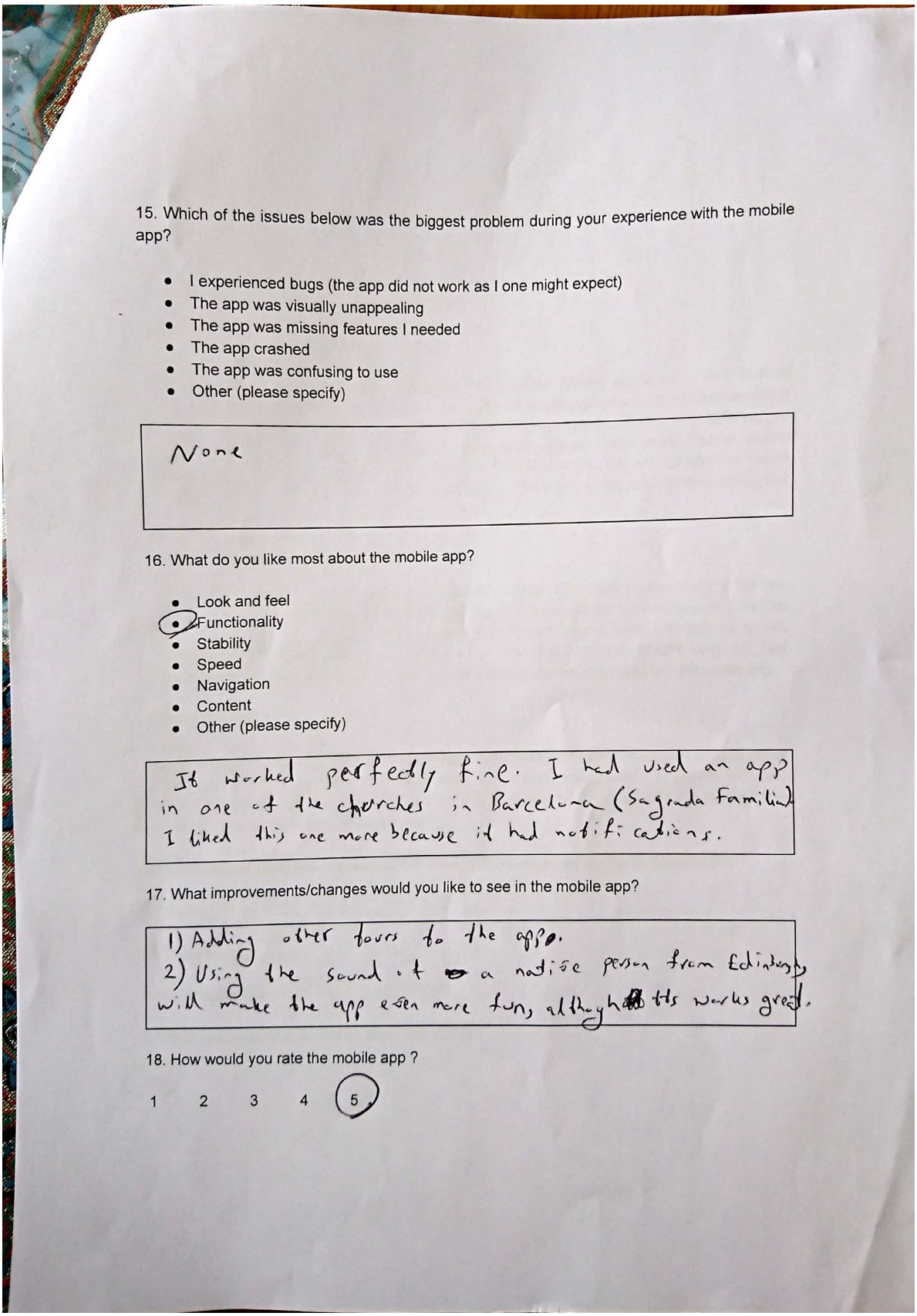
13. Notifications made exploring the city easier

Strongly Disagree Disagree Neutral Agree (Strongly Agree)

14. Which of the following did you find more convenient in guiding you through the tour

- (The directions on the map)
- The situational awareness information suggested at the end of each story (for example, "carry on down the hill")

Figure 44. User 8, page 2.



15. Which of the issues below was the biggest problem during your experience with the mobile app?

- I experienced bugs (the app did not work as I one might expect)
- The app was visually unappealing
- The app was missing features I needed
- The app crashed
- The app was confusing to use
- Other (please specify)

None

16. What do you like most about the mobile app?

- Look and feel
- Functionality
- Stability
- Speed
- Navigation
- Content
- Other (please specify)

It worked perfectly fine. I had used an app in one of the churches in Barcelona (Sagrada Familia) I liked this one more because it had notifications.

17. What improvements/changes would you like to see in the mobile app?

1) Adding other tours to the app.
2) Using the sound of a native person from Edinburgh, will make the app even more fun, although this works great.

18. How would you rate the mobile app ?

1 2 3 4 5

Figure 45. User 8, page 3.