WhichPrep! - Tech Guide

Table of Contents

1. Introduction

1.1 Overview

WhichPrep! is an educational application developed in Android. It aims to aid the user, typically a non-native English speaker in their use of prepositional words in sentences. This is the main area that people learning the English language have difficulty with. The app tests the user by presenting them with sentences missing the prepositional word and the user must pick the correct word. The sentences used are scraped from the web, the idea being that the user will be tested using real world text from articles, blogs etc. The app will provide feedback and identify weaknesses in order to help the user develop their level of English.

1.2 Glossary

***Android Studio*:** Android Studio provides the fastest tools for building apps on every type of Android device.

***Java*:** Java is a general-purpose computer programming language that is concurrent, class-based, object-oriented, and specifically designed to have as few implementation dependencies as possible.

XML: Extensible Markup Language

JSoup: Java HTML parser library. Supports extracting and manipulating data using DOM, CSS, and jquery methods.

PhoneGap: Development tool for creating cross-platform mobile applications using HTML, CSS, & Javascript.

Web Scraping: This is a technique used in computing for collecting and extracting information from websites.

SQLite: SQLite is an in-process library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine.

1.3 Motivation

The project is based on an idea put forward by Jennifer Foster of The School of Computing, DCU, as a final year project.

On a personal level, i chose to do this project for a number of reasons. For a start, it gave me the opportunity to learn something new. My project last year was a web-based application using the likes of HTML, CSS, and Javascript. This project idea allowed me to use new technology, such as Android Studio, XML, SQLite, JSoup among others.

Secondly, i liked the idea of developing a language learning app. As someone who learnt other languages in the past, it was often the little words i.e the prepositions, where i accumulated the most errors. I liked the idea of an application that helped to develop this skill.

1.4 Research

Web-Scraping & JSoup

One of the main aspects of the application is its ability to get content from the web in order to test the user. I had little experience up to this point in web scraping. I was first exposed to it last semester for the Search Technologies module, but this was very primitive. I researched online and with others and was recommended to use JSoup. This suited my project as JSoup is written in Java as are Android Studio projects. Implementing Jsoup into my project. I started writing simple Jsoup programs and felt that it was the right choice for the project.

Prepositions

In order to make a good application, i needed to do plenty of research into what prepositions are, when they are used, how many of them there is etc. I spoke with Jennifer Foster before the project proposal and she pointed me in the right direction. I looked at papers online which spoke about why prepositions are often the biggest source language errors for non-native speakers.

I also had to learn the rules of prepositions. There are a few very general rules for the use of prepositions, i.e prepositions must be followed by noun, but not always, a preposition can go on the end of a sentence, but not always, certain prepositions must follow certain words in order to make sense etc.

Prepositions are unlike other types of words in that there are no concrete rules that define them, this is why non-native english speakers have trouble with them.

Agile Methods

I started work on this project prior the start of semester two, when we commenced the software principles module. Up until then, the development of my project had no proper structure or life-cycle in place. Luckily however, i had not implemented a huge amount of my project before Semester Two, and so in the first few weeks we learned about the different approaches to software development. I adopted an ‘incremental’ style approach, where i would build the project early, evaluate, and then add functionality in increments.

2. System Architecture

2.1 Design

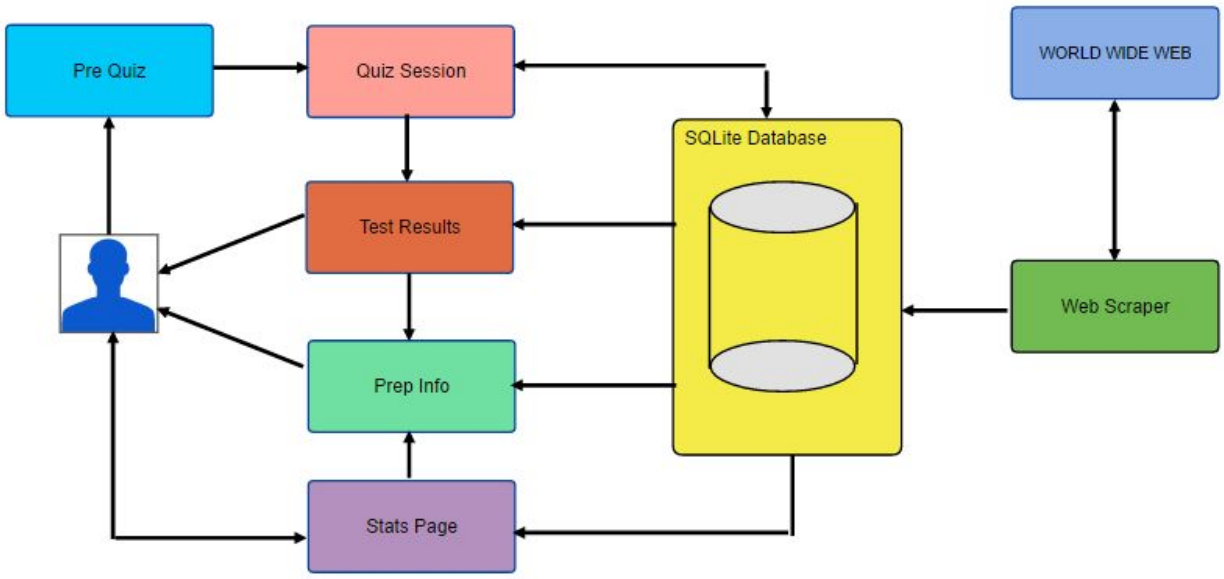


Figure 2.1

Figure 2.1 above shows an overview of the system architecture and its main components.

Web Scraper

Scrapes data from www.theguardian.co.uk , formats it and stores into the database. This runs periodically in the background once every few days, in order to collect new test data to be used in the quiz.

SQLite Database

The applications database, stored in local memory. It consists of two tables. The first table stores the prepositions and relevant information on them. The second table stores the test sentences scraped from the web.

Pre-Quiz

User can specify the test conditions here, prior to starting the test. Can set difficulty, test type, and time per question.

Quiz Session

Retrieves 10 sentences from the database and tests the user. Each question is on a timer. On completion of each sentence, the result is stored in the preposition table in the database. On completion of a test, user is brought to results page.

Results

Shows the user the results of the previous test by displaying buttons for each question. The colour of the button will be green if the user got that question right, or red if they got it wrong.

Stats Page

Shows a user their progress. Splits prepositions into 3 fields: weak, fair, and strong. These are calculated by checking the number of times user has got them correct and incorrect. They give the user an indication of which prepositions they are struggling with,. User can click on a preposition to show relevant info.

Prep Info

This page shows information on the preposition, providing its meaning, giving examples of how/when it's used. It will also show the number of times it has been asked, along with the number of times the user has chosen it correctly.

3. High Level Design

3.1 Web Scraper

3.1.1 Jsoup Parser

The Web Scraper works in the following way. It starts with a basic url, www.theguardian.co.uk

A number of set topics have been set, so a loop is created to append each topic to the url. In each of those loops, the date (- 1) gets appended onto the url. This results in the following url.

www.theguardian.co.uk/football/2016/may/23/. At this point the application runs the *getLinks()* method, which collects all url's (which will be articles) from the above page and returns them in an ArrayList. This then gets passed to the *getArticle()* method, which collects all text from each article, splits it into sentences, and these sentences get added to the database, depending on whether or not a prepositional word/phrase exists in that sentence. Below is some sample code highlighting the above process.

```
public void splitString(String text){
    if(!(text == null)) {
        String[] parts = text.split("\\. ");
        for (String sentence : parts) {
            if (numWords(sentence) < 18 && numWords(sentence) > 6)
                dbHandler.getSentence(sentence);
        }
    }
}


public static String getArticle(String url){
    try
    {
        Document doc = Jsoup.connect(url).get();
        Elements article = doc.select("div[class=content__article-body from-content-api js-article__body");
        return article.text();
    }
    catch(IOException e)
    {
        e.printStackTrace();
    }
    return null;
}

public void getSentences(){
    for(String link: getLinks()) {
        splitString(getArticle(link));
    }
}
```

3.1.2 Periodic Async Web Scraper

As the Web Scraper is quite a time consuming process, the application runs it as an asynchronous task, meaning it can run in the background while the user still uses the app. Further, rather than have the webscraper run every time the app is opened, it should only run periodically every few days. A Broadcast Receiver was implemented for this, and an Alarm which sets the time for the process to run. This can run in the background without needing the app to be open. On the very first run of the application, this task is called straight away. After that, it only runs at a specific time every few days. Below is some code which implements this. Figure 3.1 shows the first run of the app, which starts the alarm. Figure 3.1.2 shows how the alarm is set.

```
@Override
protected void onResume() {
    super.onResume();

    if (prefs.getBoolean("firstrun", true)) {
        Intent alarmIntent = new Intent(this, AlarmReceiver.class);
        pendingIntent = PendingIntent.getBroadcast(this, 0, alarmIntent, 0);
        start();
        prefs.edit().putBoolean("firstrun", false).apply();
    }
}
```

Fig 3.1.1

```
public void start() {
    AlarmManager manager = (AlarmManager) getSystemService(Context.ALARM_SERVICE);

    /* Set the alarm to start at 12:00 AM */
    Calendar calendar = Calendar.getInstance();
    calendar.setTimeInMillis(System.currentTimeMillis());
    calendar.set(Calendar.HOUR_OF_DAY, 0);
    calendar.set(Calendar.MINUTE, 0);

    manager.setRepeating(AlarmManager.RTC_WAKEUP, calendar.getTimeInMillis(),
            AlarmManager.INTERVAL_DAY * 7, pendingIntent);
}
```

Fig 3.1.2

3.2 SQLite

3.2.1 Preposition table

The application makes use of an sqlite database, which is stored in local memory on the user device. It consists of two tables. The first table stores the prepositions to be used in the test. Columns include the level, description, example, num_used, and num_correct. This table gets populated on the first run of the application, to add 120+ prepositions. Figure 3.2.1 below is an example.

```
dbHandler.addPrep(new Preposition("aboard", 2, "à bord", "a bordo", "auf, an Bord", "on or into (a ship, aircraft, train, or other vehicle).", "The captain went aboard the aircraft", 0, 0));
```

Figure 3.2.1

3.2.2 Preposition class

The preposition class is what is used to store prepositions into the database. Figure 3.2.2 below is the constructor for the preposition class. This class implement getter and setter methods for each of the variables below.

```
public Preposition(String prepname, int level, String french, String spanish, String german, String description, String example, int num_used, int num_correct) {
    this._prepname  = prepname;
    this._level     = level;
    this._french    = french;
    this._spanish   = spanish;
    this._german    = german;
    this._description = description;
    this._example = example;
    this._num_used = num_used;
    this._num_correct = num_correct;

}
```

Figure 3.2.2

3.2.3 Test Sentence table

Stores the sentences scraped from the web to be used in the applications quiz function. Column's include the sentence, prep, user_choice, question_num. The last two columns are only populated if the sentence is being used in a quiz, once the quiz finishes they are reset.

3.2.4 Test Sentence class

The TestSentence class is what is used to store sentences into the database. The class is similar to the preposition class above. It implements the same methods, the only difference is the variables used for constructor.

3.3 Pre Quiz

This is the page where the user specifies the test conditions. It extends the android ‘activity’ class, which is basically a UI on the screen. There are four aspects to the Pre Quiz class.

1. Level - Specifies the test level i.e what the user is tested on.
    - The front end consists of four radio buttons, levels 1-4.
    - The back end records the user’s choice as an int and stores in a bundle(more on this later)

2. Answer Suggestions - Specifies the type of test the user will take, whether they choose answer from a list or type/say answer.

    - The front consists of a simple toggle button, ON and OFF
    - The back end records the choice and store in the bundle.

3. Time per question - Specifies the time for each question. When it runs out and the user has not made a choice, the quiz will automatically go to the next question, and the previous question will be marked as wrong.

    - The front end consists of a slider, set to 30 seconds. It can be set to 20-60 seconds by the user.
    - The back end records the user’s choice and sends to the bundle.

4. Start Quiz - This button is used to start a quiz. This is where the ‘Bundle’ mentioned above comes into play. The bundle is basically a multi-type array passed to an ‘intent’ which is used to pass data between activities in android. In this case the bundle contains an integer value(level), a boolean(answer suggestions) etc. These will be used in the Quiz page activity. Figure 3.3.1 below shows the method called when the start button is clicked.

```
public void startQuiz(View view){
    Intent i;
    if(answerSuggestions.equals("true"))
        i = new Intent(this, QuizActivity.class);
    else
        i = new Intent(this, AltQuizActivity.class);
    Bundle bundle = new Bundle();
    bundle.putInt("level", levelSelected);
    bundle.putString("answerSuggestions", answerSuggestions);
    bundle.putString("time", time);
    i.putExtras(bundle);
    dbHandler.resetSentences();
    startActivity(i);
}
```

Figure 3.3.1

3.4 Quiz Session

The quiz activity accesses the values of the bundle from the previous activity(pre quiz) to be used in creating the test environment.

3.4.1 General

- Front end - This page shows the question number at the top of the page. Below this is a textbox containing the test question. The preposition has been replaced with '___'. The application retrieves this sentence from the database,sets it's 'used column to 1, and sets the question_num column to the number of the question. The first ensures this sentence won't be used again in this test, the second is used for the results activity later. Figure 3.4.1 below shows a small snippet of how this is done.

```
public String prepareSentence(int level) {
    String sentence = "";
    SQLiteDatabase db = getWritableDatabase();
    String query = "SELECT * FROM " + TABLE_TESTSENTENCE + " WHERE " + COLUMN_LEVEL + "=" + level + " AND " + COLUMN_USED + "=0 ORDER BY RANDOM() LIMIT 1";
    Cursor cursor = db.rawQuery(query, null);
    cursor.moveToFirst();
    while (!cursor.isAfterLast()) {
        if (cursor.getString(cursor.getColumnIndex("sentence")) != null) {
            sentence = cursor.getString(cursor.getColumnIndex("sentence"));
            db.execSQL("UPDATE " + TABLE_TESTSENTENCE + " SET " + COLUMN_USED + "=1 WHERE " + COLUMN_SENTENCE + "='" + sentence + "'");
            break;
        }
        cursor.moveToNext();
    }

    cursor.close();
    db.close();
    return sentence;
}
```

Figure 3.4.1.1

The timer is shown in the bottom left corner and the next button in the bottom right. This page makes use of the *activityCounter()* method. This method is external the the quiz class. At each change of question, the quiz activity page gets reloaded. However it references general MyApplication class. The *activityCounter()* method calls the *getCount()* in MyApplication, which basically tells the quiz activity which question it should be on. When it reaches the tenth question and *activityCounter()* is called, it knows to end the test and progress to the results activity. This can be seen in figure 3.4.1.2 below.

```
public void activityCounter(int count){
    if(count != 10) {
        Intent intent = getIntent();
        finish();
        startActivity(intent);
    }
    else {
        startActivity(new Intent(QuizActivity.this, ResultsActivity.class));
    }
}
```

Figure 3.4.1.2

3.4.2 Test type 1

If the user selected answer suggestions to 'ON' on the previous page, they will be presented with a list of three prepositions, one of which is the correct one. Clicking on any prep will highlight it. The user can change their choice by clicking on a different option in the list. Once the user is happy with their choice, they can press the next button. This does the following:

- Adds the user choice to the database in the row for the correct preposition.
- Adds the question_num in same row, to be used for result activity later.
- Saves the result in the preposition table under the num_used and num_correct columns.
- Calls *activityCounter()* method.

The following is a snippet of one of the above methods.

```
public void saveResult(String prep, String userChoice){
    SQLiteDatabase db = getWritableDatabase();
    db.execSQL("UPDATE " + TABLE_PREPOSITION + " SET " + COLUMN_NUM_USED + "= " + COLUMN_NUM_USED + " + " + 1 + " WHERE " + COLUMN_PREPNAME + "='" + prep + "'");
    if(userChoice.equals(prep))
        db.execSQL("UPDATE " + TABLE_PREPOSITION + " SET " + COLUMN_NUM_CORRECT + "= " + COLUMN_NUM_CORRECT + " + " + 1 + " WHERE " + COLUMN_PREPNAME + "='" + prep + "'");
    db.close();
}
```

Figure 3.4.2.1

3.4.3 Test type 2

If the user selected answer suggestions to 'OFF' on the previous page, they will be presented with just a text field, where they must type in the word they think fits the sentence. Alternatively, they can click the mic button, which starts a speech recognizer. When it clicked it plays a sound and the user can say a word. The interpreted word is then placed in the text field. To do this i had to implement the RecognitionListener class. Below is a snippet of what happens when the button is clicked.

```
btnSpeak.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        if (v.getId() == R.id.btnSpeak)
        {
            countdown.cancel();
            Intent intent = new Intent(RecognizerIntent.ACTION_RECOGNIZE_SPEECH);
            intent.putExtra(RecognizerIntent.EXTRA_LANGUAGE_MODEL,RecognizerIntent.LANGUAGE_MODEL_FREE_FORM);
            intent.putExtra(RecognizerIntent.EXTRA_CALLING_PACKAGE, getClass().getPackage().getName());

            intent.putExtra(RecognizerIntent.EXTRA_MAX_RESULTS, 3);
            sr.startListening(intent);
        }
    }
});
```

Figure 3.4.3.1

3.5 Results

3.5.1 Main Results

This page consists of 10 buttons, each button for one of the test questions. Each button calls the the *getResult()* method, which takes the question number as a parameter and queries the sentence table to find what the user's choice was. If true(user was correct), the button will be green. It will be red otherwise. This can be seen in the below code snips.

```
if(!dbhandler.getResult(1))
    q1.setBackgroundColor(Color.parseColor("#e74c3c"));
if(!dbhandler.getResult(2))
    q2.setBackgroundColor(Color.parseColor("#e74c3c"));
```

Figure 3.5.1.1

```
public boolean getResult(int num){
    String prepname = "";
    String userChoice = "";
    SQLiteDatabase db = getReadableDatabase();
    String query = "SELECT " + COLUMN_PREPNAME + ", " + COLUMN_CHOICE + " FROM " + TABLE_TESTSENTENCE + " WHERE " + COLUMN_QUESTIONNUM + "=" + num + "";

    Cursor cursor = db.rawQuery(query, null);
    cursor.moveToFirst();
    while (!cursor.isAfterLast()) {
        if (cursor.getString(cursor.getColumnIndex("prepname")) != null) {
            prepname = cursor.getString(cursor.getColumnIndex("prepname")).substring(1);
        }
        if (cursor.getString(cursor.getColumnIndex("choice")) != null) {
            userChoice = cursor.getString(cursor.getColumnIndex("choice"));
        }
        cursor.moveToNext();
    }

    cursor.close();
    db.close();
    return prepname.equals(userChoice);
}
```

Figure 3.5.1.2

3.5.2 Specific Question Results

Clicking on any of the above questions will take you to a post quiz view of that question. It is similar to the normal quiz page in that it shows the question number and the question itself. The question number is passed in a bundle from the results page to this page. It then uses this number to query the database and return an array containing the results..

Below this shows the correct preposition for that sentence, highlighted green and the user's choice below(highlighted green if correct, red if wrong). The screenshot below

```
dbhandler = new MyDbHandler(this, null, null, 0);
bundle = getIntent().getExtras();
questionNum = bundle.getInt("questionNum");

results = dbhandler.displayResult(questionNum);
correctPrep = (Button) findViewById(R.id.correctPrep);
userPrep = (Button) findViewById(R.id.userPrep);
correct =  results.get(1).substring(0 ,results.get(1).length()-1);
userChoice = results.get(2);

if(results.get(1).equals(results.get(2))){
    userChoice = correct;
}
```

Figure 3.5.2.

Clicking on either of the prepositions will take the user to a screen showing information on that preposition. This is described in section 3.6 below.

3.6 Stats page

3.6.1 Preposition stats

The page shows the user a number of subpages, each subpage shows user statistics e.g Level 1 Stats, Level 2 Stats. Clicking on one of these will bring the user to a page showing statistics on the prepositions for that level. It does this by passing the 'level' to the new activity using a bundle intent.

An example for the 'Level 1 Stats' can be seen below in figure 3.6.1.1

```
public void showLevel1(View view){
    bundle.putInt("level", 1);
    i.putExtras(bundle);
    startActivity(i);
}
```

Figure 3.6.1.1

The above opens the main stats page, and shows the user's weak, fair, and strong prepositions. The preposition are sorted by comparing the number of times a certain preposition has been used in a sentence, to the number of times it has been answered correctly. It does this by calling the *getPreps()* method, which takes three integer parameters, low, high and level. Level is in relation to the preposition, low and high define the percentage

range of the prepositions ‘num_correct’ over ‘num_used’ columns. The following is a small snippet from the method that retrieves the prepositions.

```
if(used != 0 && used >= 10)
    percentage = (correct*100) / used;
if(percentage != -1) {
    if (percentage > low && percentage <= high) {
        prepositionList.add(prepname);
    }
}
```

Figure 3.6.1.2

Only prepositions which have been used 10 or more times will be shown in these lists, as anything below this won’t truly reflect the user’s ‘correctness’ on them.

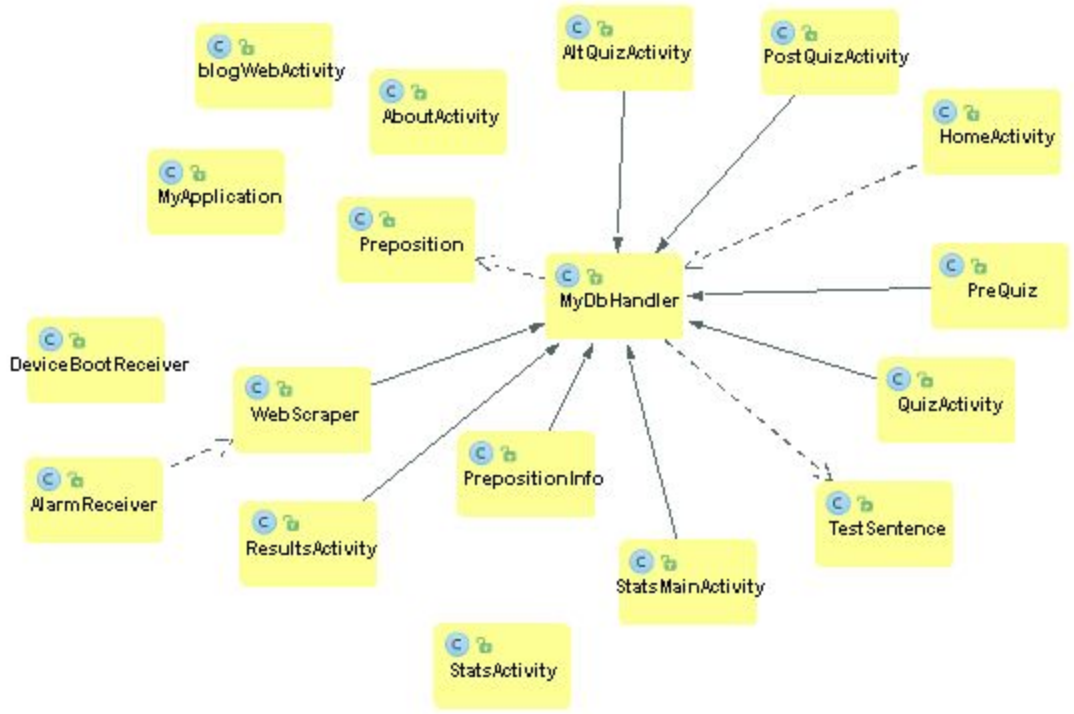
3.6.2 Practice Weak Preps

A fifth option on the stats page, entitled ‘Practice Weak Prepositions’, will start the quiz activity again. This time however, the user will only be tested on the prepositions which they have been getting wrong more than 50% of the time. This is done by modifying the *prepareSentence()* to take a boolean as a parameter. This boolean will be set to true when this method is called from ‘Practice Weak Preps’. It lets the system know to only use sentences with prepositions that the user is having difficulty with. A small snippet of the updated *prepareSentences()* method can be seen below.

```
if (cursor.getString(cursor.getColumnIndex("prepname")) != null) {
    prepname = cursor.getString(cursor.getColumnIndex("prepname"));
    if(weakPreps.contains(prepname)){
        if (cursor.getString(cursor.getColumnIndex("sentence")) != null) {
            sentence = cursor.getString(cursor.getColumnIndex("sentence"));
            db.execSQL("UPDATE " + TABLE_TESTSENTENCE + " SET " + COLUMN_USED + "=1 WHERE " + COLUMN_SENTENCE + "='" + sentence + "'");
            break;
        }
    }
}
```

Figure 3.6.2.1

3.7 Class Diagram



4. Problems & Resolution

4.1 PhoneGap Issues

Problem

When i initially began to develop the project, i was using PhoneGap, a tool for developing cross platform mobile applications using HTML, CSS and JavaScript. My plan was to build the front end first, and add functionality later, something which in hindsight was very naive of me. When i started to add functionality, i started to run into issues. For example, i spent quite a number of days trying to set up a database alone. Phonegap requires an external plugin to use for its database. I followed tutorials online and took all the necessary steps but could not get it to work

Solution

I spoke to my supervisor about the issue. We talked about PhoneGap and why i was using it, and we decided then to cut my losses and move on to using android studio. As the project is a proof of concept, it was not overly important to have a working cross-platform application.

4.2 WebScraper - URL Types

Problem

When i first wrote the webscraper, i wrote it separate to my project, in eclipse. It was able to compile and run, printing out the sentences it had scraped. Every so often in the cmd window however, an error would pop up, indicating an out of bounds error. I didn't pay too much attention to this as the program still ran and i was getting the results i wanted. When i imported the Jsoup code into my android project, it caused the app to crash. This forced me into looking into this error further.

Solution

After much troubleshooting, i discovered why the error was occurring. The scraper looked at multiple links to articles in order to get its text. However in some cases, the pages that these links referenced had different source code than the others. When the program went looking for a specific element on the page and didn't find it, it threw the error. Types of links included links to galleries, video etc. To solve the problem i filtered out these url types by checking for their existence in the url before running the parser.

4.3 Speed of WebScraper

Problem

With the webscraper up and running, i started to test it by gathering content on large numbers of articles. It worked fine, but was very slow. This was not going to work in the apps favour. The user would quickly lose interest if they had to wait before be able to take the test.

Solution

I implemented the webscraper as an asynchronous task. This meant that the user could use the app while the scraper ran in the background.

Further, i didn't want the webscraper to be run everytime the user opened the app as it was unnecessary and may needlessly use up data. To solve this, i implemented a broadcast receiver and alarm manager. Itis in the broadcast receiver that the webscraping is done, and the alarm is used to call the broadcast receiver periodically once every few days.

4.4 Unbalanced Frequency of Prepositions in Quiz

Problem

After implementing the quiz functionality and using it for a while, i noticed a significant issue. In a test of 10 questions, there was no limit to the amount of times a certain preposition was the answer for numerous questions. Obviously some prepositions are a lot more common than others. For example the word ‘it’ appeared extremely frequently in the tests.

Solution

To solve this issue, i needed to ensure that a certain preposition could only be a solution to one of the questions in the test, which gave the test and the app better coverage of all prepositions. I implemented a check on the method which retrieved the sentences to be used. Once a sentence was added,

4.5 Speech Recognition Issues

Problem

After spending some time on trying to implement speech recognition, i was able to get it working. The problem was, it didn’t work very well. It misinterpreted many of the words I was trying to say. For example, when i said ‘to’, it result in placing ‘2’ in the text field, saying ‘than’ always showed up as ‘van’, ‘with’ sometimes gave me ‘width’.

Solution

To solve this, i needed to research how the speech recogniser had been implemented. I had it set up to only return one word. However after looking online on stackoverflow and other websites, i discovered that the speech recogniser can return an array of all probable words that the said word was. I changed this aspect of my code. Next, i iterated through the returned array, checking if the word matched any of the prepositions in the database. If it was a match, it would return this word and place it in the text field, if not, it would print nothing. This effectively reduced the grammar that the speech recogniser could interpret.

5. Results

5.1 Results

5.1.1 Functional Specification vs Finished Application

- In my functional spec i stated that i was going to use PhoneGap in order to build a cross platform app, however due to difficulties with this technology, and lack of help and

documentation online, it was decided to develop using Android Studio. I also felt it was probably better to use as it would be better to build a good, reliable application on one platform, than an un-reliable cross platform one.

- The functional spec alluded to the application having a Login/Register system. However as i began to develop the system, designing it so that the database was stored in local memory, i didn't feel there to be any need for registering users. The device saves all the data for the user and it is personal to them. The only issue would be if multiple users were using the same device, which is extremely rare.
- In the functional specification, i mentioned that the user could specify their areas of interest on the application, and it would then use this information to scrape specific data. After testing the functioning app however, i realised there was little to no advantage of having this, as most of the sentences scraped were too general, regardless of what category of article they were scraped from.

5.1.2 User Acceptance Tests

Below are a couple of the user acceptance tests used for the application. There were over 15 in total, so i won't include all of them here.

Test Case 5

Name: JSoup Webscraping	Design date: 26/3/2016
Test Priority: Very High	Execution date: 26/3/2016
Designed by: Conor McGovern	
Executed by: Conor McGovern	

Test Purpose

The purpose of this test is to ensure that the application has the ability to scrape content from the web to be used in the quiz section of the app. This test will aim to scrape sentences of a particular length from one article only, taken from www.theguardiam.co.uk

Pre conditions:

- Jsoup jar file added to android project.
- Internet connection present.
- Jsoup imports added to main java program

- Listview set up in app, which is to be populated by web content.

- Asynchronous task implemented, where webscraping code takes place. Async task allows for code to run in the background, while app runs as normal.

Step	Action	Expected System Response	Pass/Fail	Comment
1	Build and run the application on device/ emulator	Application compiles and builds successfully, and starts on the device. Listview is populated with text taken from the web.	Pass	There is a slight pause before the sentences appear in the app.

Notes:

While this test was successful, it will need to be revised and redone to test more complicated versions of the scraper which can take text from multiple articles.

Test Case 12

Name: Quiz Page

Design date: 21/4/2016

Test Priority: Very High

Execution date: 21/4/2016

Designed by: Conor McGovern

Executed by: Conor McGovern

Test Purpose

The purpose of this test is to ensure that the application can perform a test on a particular sentence for the user.

Pre conditions:

- Sentences exist in database.

- Test conditions from pre-quiz have been recorded.

Step	Action	Expected System Response	Pass/Fail	Comment

1	From pre-quiz page, press ‘start quiz’ button.	Application opens a new page, showing a sentence, a list of three prepositions, a timer, and a ‘next’ button.	Pass	
2	Press ‘next’ button	Message appears, telling user they must choose a preposition from the list	Pass	
3	Allow timer to run out	The app will jump to the next question	Pass	If the user has not selected a prep, the question is marked as incorrect.
4	Click on any prep from the list	The preposition is highlighted, indicating the users choice.	Pass	

6. Future Work

As this application was built for college purposes only, my intention is not to release or deploy. It was designed more for a proof of concept. Also, permission was required from theguardian.co.uk to use their content in my application(proof of this permission can be seen in Ethical Approval documentation), so there may be issues further down the line if it was released and used publicly.

Hypothetically speaking, if i was to continue on with this project, i would try to implement more elements of machine learning into the application, such as the app suggesting to the user the test conditions, introduce a levelling system, where user must show proficiency in one section before moving on to a harder section etc.