ДАТА: 05.01.2023

Технически Университет – София

ФКСТ

Компютърно и софтуерно инженерство

Курсов проект

Програмиране за мобилни устройства

Weather application

ПРЕДСТАВЕНО ОТ:
ВЕСЕЛИН ПЕТРАНЧЕВ
121220069
44ГР.

РЪКОВОДИТЕЛ:
проф. Огнян Наков

СЪДЪРЖАНИЕ

Увод

Мобилните приложения за времето са едни от най-полезните инструменти за съвременния човек. Те ни позволяват да следим промените в атмосферните условия и да получаваме прогнози за времето за определен период от време. Въпреки това, много от тези приложения имат своите проблеми, като например неточни прогнози, бавно обновяване на информацията, затруднен достъп до специфични данни и т.н.

Ако имате мобилно приложение за времето, което среща тези проблеми, то това може да доведе до недоволство и отказ от използването му. Това може да доведе до загуба на потребители и да намали успеха на приложението.

Затова е важно да се реши този проблем, като се осигури точна и навременна информация, бързо обновяване на данните и удобен достъп до специфични функции. Така потребителите ще могат да използват приложението си с увереност и да се възползват от всички предимства, които то им предлага.

Анализ на съществуващи разработки

I. **AccuWeather -** това приложение е известно със своята точност на прогнозите за времето, като използва интелигентни алгоритми за анализ на данните от много източници. Освен това, това приложение има и много други полезни функции, като например информация за UV индекса и националните прегледи на времето в САЩ. Отрицателната страна на AccuWeather е, че приложението има много реклами, които може да бъдат дразнещи за потребителите.

II. **The Weather Channel -** това приложение е много популярно сред потребителите, тъй като предоставя информация за времето по целия свят, а не само в САЩ. Приложението предлага точни и навременни прогнози за времето, както и много полезни функции, като например информация за качеството на въздуха. Отрицателната страна на The Weather Channel е, че приложението може да бъде малко бавно при зареждане на информацията, което може да бъде дразнещо за потребителите.

III. **Dark Sky** - това е приложение, което е известно със своите прогнози за времето в реално време. Приложението използва GPS данни, за да предостави точни и навременни прогнози за времето на местонахождението на потребителя. Освен това, Dark Sky има и много други функции, като например информация за превръзките на небето и радар за проследяване на близките бури. Отрицателната страна на Dark Sky е, че приложението не предоставя много информация за времето за по-големи периоди от време, като например за седмици или месеци.

Проектиране

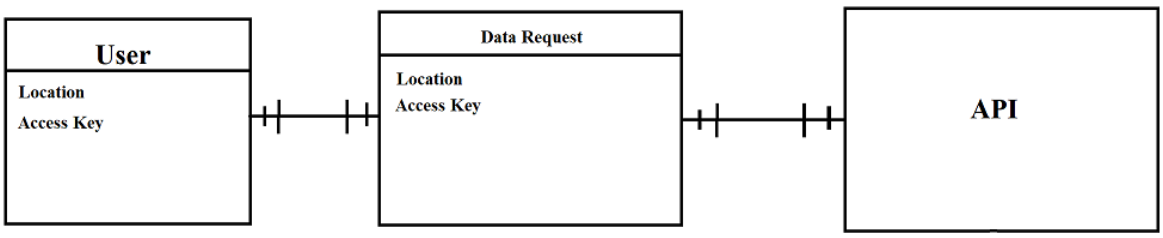
I. Кой ще използва продукта?

Целевата аудитория за приложението за времето са хората, които имат нужда от редовна информация за прогнозата за времето в даден регион. Това могат да бъдат хора, които пътуват много и имат нужда от информация за времето в различни градове, спортисти, които имат нужда от точна информация за прогнозата за времето за своите тренировки, и други.

II. Какви данни ще се използват?

Приложението за времето трябва да може да използва данни за текущите метеорологични условия и да предоставя прогноза за времето за близките дни. В момента това ще бъде за текущото време понеже, за да взема повече данни ще трябва да се плаща за абонамент. Данните, които ще се ползват са предоставени от https://openweathermap.org/ , като това е API, който бива достъпван на https://api.openweathermap.org/data/2.5/weather?q=$city&units=metric&appid=$api

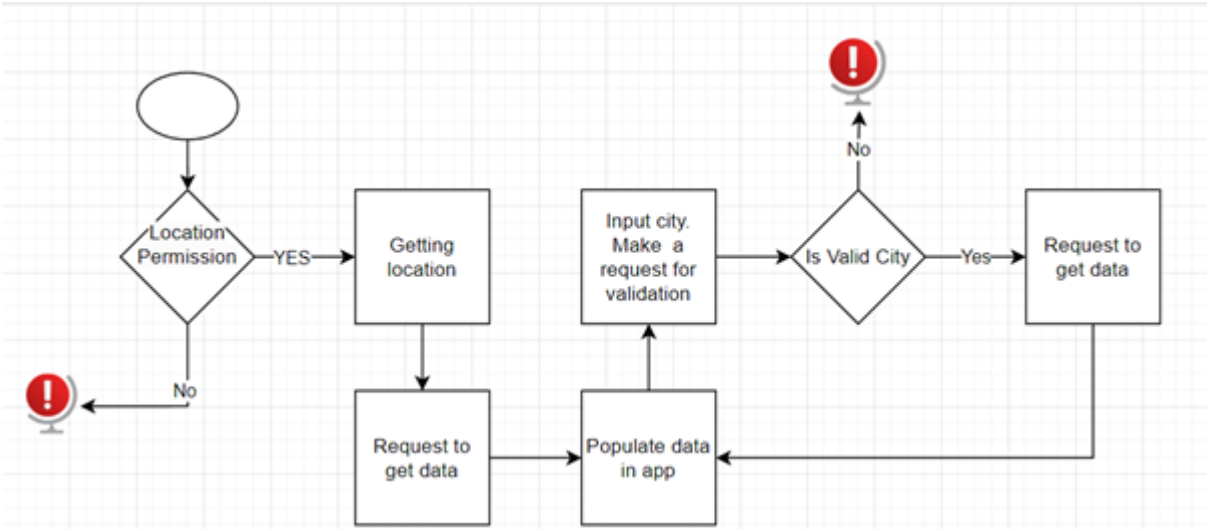
Където $city => е името на града, на който искам да достъпя данните за времето, а $api е ключът за защитата, който ми позволява да направя тази заявка. Важно! Този ключ е от безплатния тестови период от 30 дена. След това се заплаща.



III. Как ще бъдат достъпени функционалностите от потребителя?

Приложението трябва да има потребителски интерфейс, който да бъде лесен за използване и да предоставя лесен и интуитивен начин за достъп до функционалностите на приложението. Навигацията в приложението трябва да бъде ясна и интуитивна, за да позволи на потребителя да намира търсената информация бързо и лесно. Освен това, приложението трябва да бъде съвместимо с различни устройства и операционни системи.

PROCESS DIAGRAM OF WORKFLOW



Реализация

Приложението работи на принципа:

1. Взима адрес, ако му бъде разрешен достъп
2. Изпраща заявка метеорологичен api, който връща нужните данни и се дисплейват.
3. Потребителят, ако иска може да въведе друг град като се проверява дали е валиден град и ако е се прави същата заявка за взимане на данните.
4. 3 точка се повтаря колкото пъти иска потребителят.
5. Има настройки като за момента можеш да смениш езика, като имаш 6 опции: Английски, Български, Испански, Френски, Немски, Руски и също така можеш да смениш фона от тъмен режим на светъл както и обратната. Тези неща се пазят в кеша на приложението, което значи, че след напускане на приложението и повторното му отваряне настройките ще се запазени.

```
override fun doInBackground(vararg params: String?): String? {
    val response:String? = try{
        URL(

"https://api.openweathermap.org/data/2.5/weather?q=$city&units=metric&appid
=$api"
        ).readText(
            Charsets.UTF_8
        )
    }catch (e: Exception){
        null
    }

    return response
}
```

Ето примерен код как се прави заявката към openweathermap API

С тази функция се проверява дали адресът е валиден

```
private suspend fun findCity(cityName: String): Pair<String?, String?> {
    val url = "https://geocode.xyz/$cityName?json=1"
    val client = OkHttpClient()
    val request = Request.Builder()
        .url(url)
        .build()
    return withContext(Dispatchers.IO) { this: CoroutineScope
        val response: Response = client.newCall(request).execute()
        val responseBody = response.body?.string()
        if (response.isSuccessful && !responseBody.isNullOrEmpty()) {
            val jsonObject = JSONObject(responseBody)
            var country = ""
            if (jsonObject.has( name: "standard")) {
                val standardObject: JSONObject = jsonObject.getJSONObject( name: "standard")
                country = standardObject.optString( name: "prov",  fallback: "")
            }
            val city = jsonObject.optString( name: "city",  fallback: "")
            Pair(city, country) ^withContext
        } else {
            null to null ^withContext
        }
    }
}
```

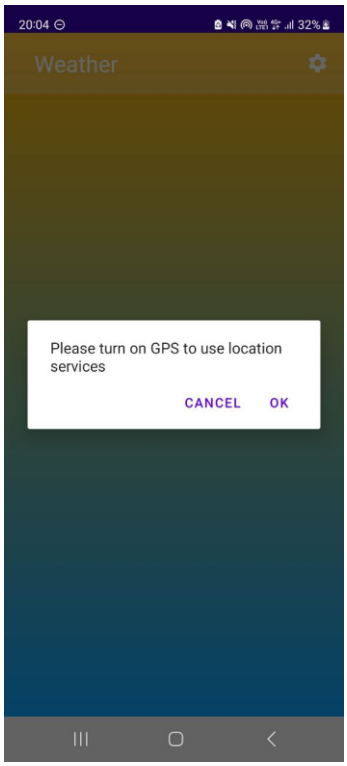
Това е json обекта от openweathermap API

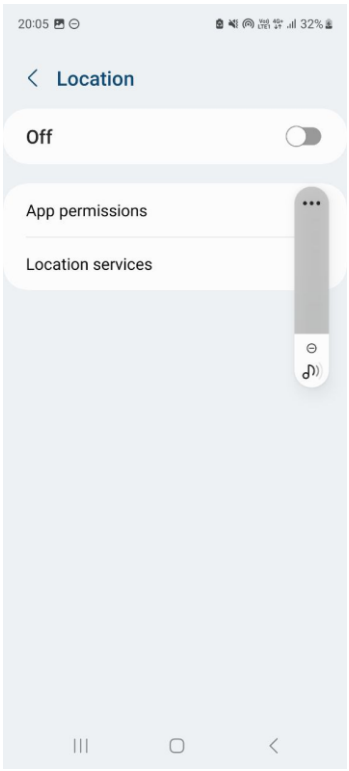

```
{
   "coord": {
      "lon": -0.1257,
      "lat": 51.5085
   },
   "weather": [
      {
         "id": 800,
         "main": "Clear",
         "description": "clear sky",
         "icon": "01d"
      }
   ],
   "base": "stations",
   "main": {
```

```
    "temp": 290.29,
    "feels_like": 289.32,
    "temp_min": 288.72,
    "temp_max": 291.79,
    "pressure": 1021,
    "humidity": 44
  },
  "visibility": 10000,
  "wind": {
    "speed": 2.57,
    "deg": 350,
    "gust": 5.14
  },
  "clouds": {
    "all": 0
  },
  "dt": 1651542001,
  "sys": {
    "type": 2,
    "id": 2019646,
    "country": "GB",
    "sunrise": 1651508792,
    "sunset": 1651565294
  },
  "timezone": 3600,
  "id": 2643743,
  "name": "London",
  "cod": 200
}
```

Потребителски ръководство (Резултати)

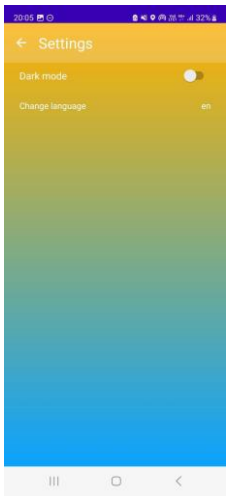
При първоначално влизане, ако приложението не може да достъпи локацията на устройството, то тогава се показва поп нотификация изискваща от потребителя да включи неговата локация и да даде достъп на приложението до нея. В противен случай то няма да може да функционира правилно. След даване на съгласие, потребителят бива навигиран в настройките на неговото устройство до настройката за пускане на локацията.

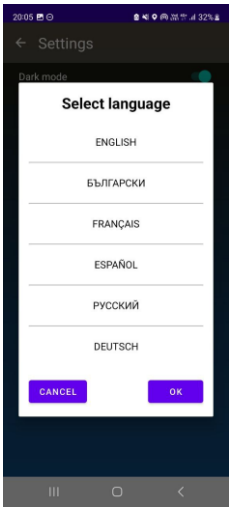




След нейното пускане трябва да се върнем ръчно до приложението, а то през това време си прави заявка към първият API чрез данните за града, в който се намира взимайки го от локацията ни. Response-а от този API връща метеорологичните данни и ние попълваме нашите полета с тях. Това се случва и когато въведем име на град в полето. Като за първа страница се визуализира следното:

От иконата горе в дясно можем да достъпим и настройките където за сега има само 2 избора: Тъмен режим и смяна на език. Изборите за език са Английски, който е по подразбиране и немски, български, френски, испански и руски.





Заключение

В заключение, след извършените анализи и проектиране на мобилното приложение за времето, може да се каже, че продуктът е готов за употреба. Той отговаря на нуждите на потребителите, като предоставя точна и актуална информация за времето, а при финансиране на приложението може да се добавят и за следващи дни. Продуктът е добре организиран и лесен за използване, с удобен потребителски интерфейс и навигация.

Литература

https://kotlinlang.org/

https://openweathermap.org/api

https://developer.android.com/courses/android-basics-kotlin/course?gclid=Cj0KCQjw6cKiBhD5ARIsAKXUdybDHjc4gwUWtgu-TN9kZELtqsXQo_nEJ5YmBEE2uCAb2xN0VWBm6PMaAklWEALw_wcB&gclsrc=aw.ds

https://geocode.xyz/

Приложение

Listener-и за смяна на езика и background-а

```
switchBackground.setOnCheckedChangeListener { _, isChecked ->
    val editor = sharedPreferencesBackgroundSwitch.edit()
    editor.putBoolean("isSwitchOn", isChecked)
    editor.apply()

    setBackground(isChecked)
}

changeLanguage.setOnClickListener { it: View!
    val languageSelectionDialog = LanguageSelectionDialogFragment()
    languageSelectionDialog.show(supportFragmentManager, tag: "languageSelectionDialog")
}
```

Функция, която проверява дали въведеният град от потребителя и реален град, изпращайки заявка до url на първата променлива.

```
private suspend fun findCity(cityName: String): Pair<String?, String?> {
    val url = "https://geocode.xyz/$cityName?json=1"
    val client = OkHttpClient()
    val request = Request.Builder()
        .url(url)
        .build()
    return withContext(Dispatchers.IO) { this: CoroutineScope
        val response: Response = client.newCall(request).execute()
        val responseBody = response.body?.string()
        if (response.isSuccessful && !responseBody.isNullOrEmpty()) {
            val jsonObject = JSONObject(responseBody)
            var country = ""
            if (jsonObject.has( name: "standard")) {
                val standardObject: JSONObject = jsonObject.getJSONObject( name: "standard")
                country = standardObject.optString( name: "prov",  fallback: "")
            }
            val city = jsonObject.optString( name: "city",  fallback: "")
            Pair(city, country) ^withContext
        } else {
            null to null ^withContext
        }
    }
}
```