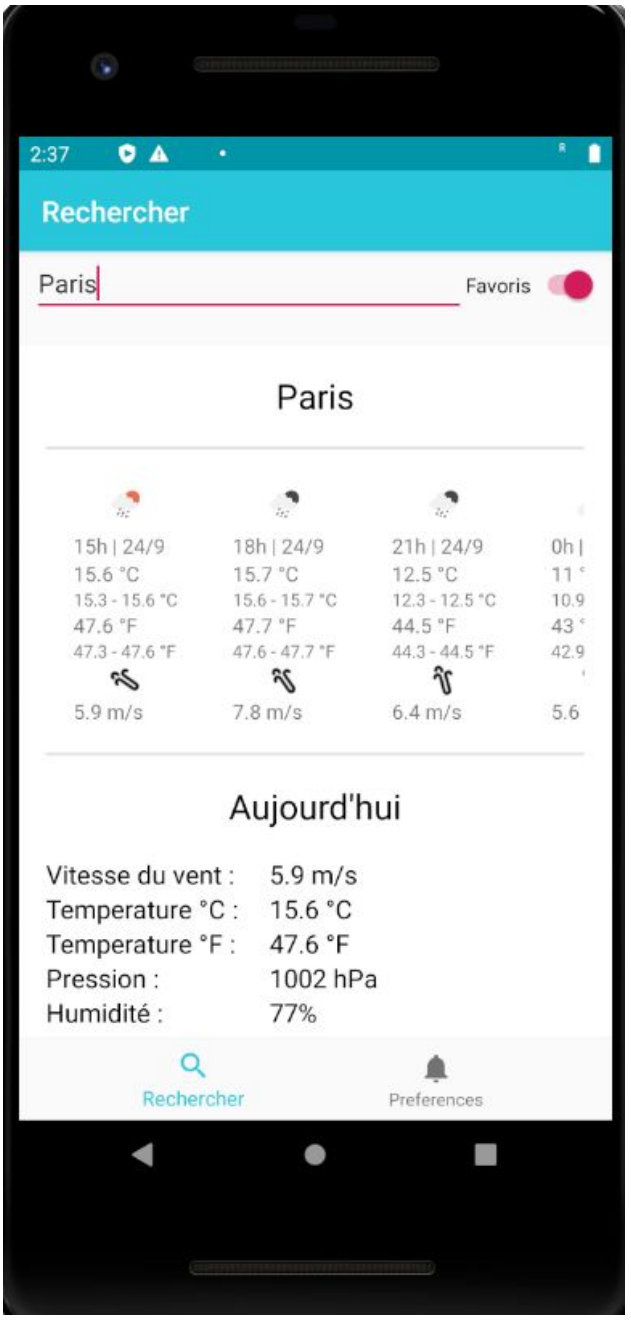
Maxime Koster
Antonin Goix

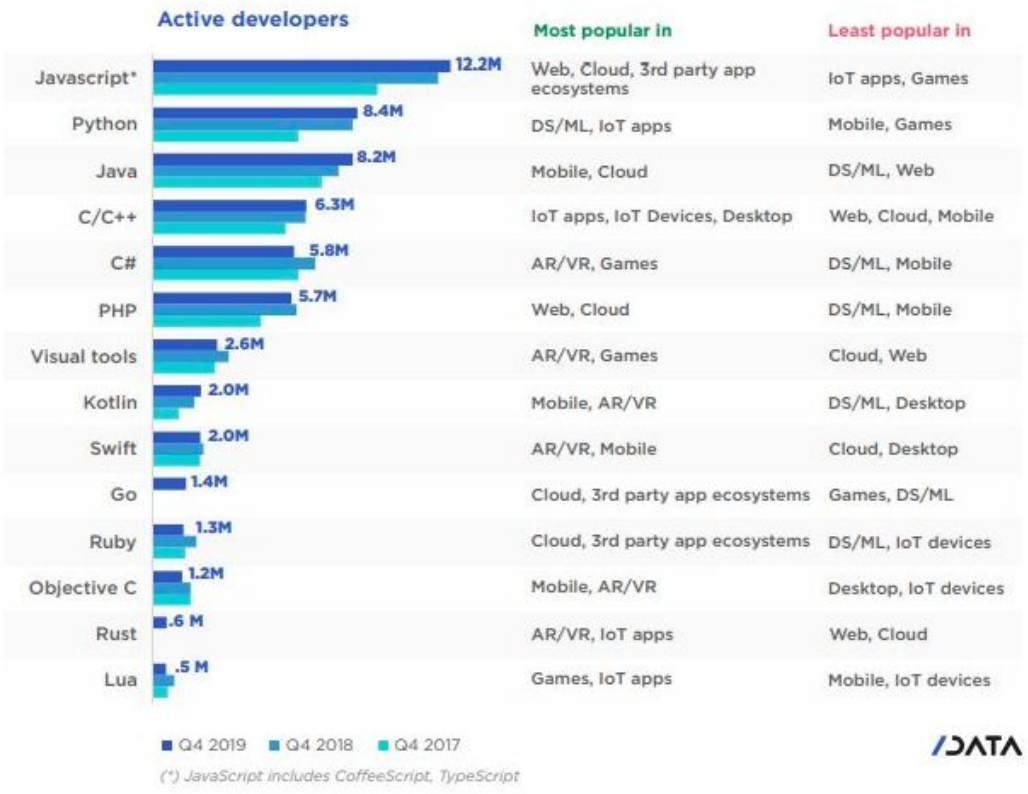
COMPTE RENDU TIC-MOB3



Sommaire

I. Choix du langage:

Nous avons choisi Kotlin pour coder cette application. Le premier objectif était d'apprendre un nouveau langage en choisissant un langage utilisé dans le monde du travail. De ce fait Kotlin était un bon choix car la demande en application native augmente de jour en jour.



Comme le montre ce schéma l'utilisation de Kotlin a doublé depuis 2017.

II. Choix des librairies:

Pour le choix des librairies malgré notre manque d'expertise dans le sujet nous avons tenté de choisir les librairies les plus pertinentes après s'être renseigné sur les différentes options possibles.

Pour les requêtes http à l'api d'OpenWeather nous avons choisi Retrofit2, la librairie la plus utilisée.
Pour récupérer les logs facilement, OkHttp3.
Pour le parsing du JSON nous avons choisi Gson.
Pour le design nous avons essayé de nous plier au règles Material Design conseillé par Google donc nous avons choisi plusieurs librairie: Material Design, Cardview, Picasso (Pour les images et le cache).
Pour les listes d'items on a bien entendu utiliser Recyclerview.

III. Initialisation du projet

1. Le projet :

Afin de commencer le projet dans les meilleures conditions nous avons opter pour un projet génère grâce à Android Studio en Kotlin, cela nous a évité d'avoir à mettre en place l'application ce qui n'est jamais plaisant pour un développeur et peut être une grosse perte de temps.
Nous sommes donc parti de cette base pour créer nos pages et nos fragments.

1. Initialisation des classes :

Grâce encore une fois à Android Studio et à sa communauté active nous avons pu facilement transformer nos json récupérés depuis l'API OpenWeather en classe Kotlin permettant de typer le retour des requêtes avec le plugin JSON to Kotlin Class.

2. Connexion avec Retrofit :

Grâce à Retrofit nous avons pu mettre en place un service api qui nous a permis de faire les requêtes sur plusieurs pages en faisant abstraction du code nécessaire à faire une requête. Ce service très pratique est couplé à un middleware injectant le token à chaque requête.
Il suffit juste d'initialiser retrofit à la création de l'écran pour tout faire fonctionner.

IV. Création de l'écran de météo

1. L'API :

Maintenant que nous avons vu les technologies avec lesquelles nous avons travaillé, nous allons pouvoir aborder la partie principale de l'application. La récupération et l'affichage des données météorologiques.
Tout d'abord nous avons mit en place un repository afin de séparer l'envois de requêtes vers l'API de OpenWeather et le code métier. Une fois le repository mit en place nous avons créée un service avec des fonctions réutilisables à travers l'application.

2. Carte principale :

Avant de créer la carte principale nous avons dû créer une page de recherche avec simplement un champs de texte pour chercher la météo d'une ville en particulier.

Une méthode a été créée dans le service afin de récupérer les informations depuis le repository en utilisant les classes générés précédemment. Le but principal de cette méthode est de faire le travail redondant de mettre toutes les infos récupérées depuis l'API dans un élément d'UI créée en parallèle. La carte principale contient les informations actuelles importantes ainsi qu'une liste de cartes plus petites montrant la météo sur 5 jours par tranche de 3 heures.

3. Cartes de prévision

Ces cartes plus compactes sont générées en même temps que la carte principale, puisque la carte principale est initialisée avec la première valeur dans la liste de prévisions météo. Pour l'occasion, un autre élément UI est créé ce qui évite à nouveau la redondance du code. Toutes les informations nécessaires sont affichées d'une façon explicite avec les unités à côté de chaque donnée (température en °C et °F, vitesse du vent en m/s … etc) ou avec des icônes pour le temps ou la direction du vent.

V. Implémentation des favoris

Les favoris n'étaient pas si facile que ça a implémenter en effet nous avons eu besoin de créer une liste d'items grâce à la RecyclerView mais aussi le besoin d'avoir de la data persistante même après un reboot de l'application. Pour ce faire nous avons utilisé les SharedPreference un système de base de données sous forme de clés-valeures embarqué. Ce système très rapide nous a permis de pouvoir mettre à jour instantanément un Switch sur la vue principale. Ce switch passe à true si le nom de la ville recherchée est déjà dans les favoris et inversement si la ville n'y est pas ce qui permet en un clique de mettre à jour la bdd persistante et donc d'avoir une liste de favoris. La liste de favoris quant à elle sous forme de Cardview permet d'afficher les détails d'une ville mais aussi supprimer des favoris la ville sélectionnée.

VII. Retouches design

1. Retouches de l'écran de météo :

Une fois le cahier des charges fonctionnel rempli, il ne restait qu'à faire des petites retouches dans l'application afin d'offrir une meilleure expérience utilisateur.
Le point principal que l'on a dû réimaginer est la carte météo. Tout d'abord, nous avons choisi une couleur penchant vers le cyan pour l'application. Cela signifie que tous les écrans ont dû respecter cette norme. Il a donc fallu changer l'écran de météo qui était en rose pâle et de même pour la police. Les contraintes de l'écran de recherche ont également dû être revues à cause de l'ajout du bouton de favoris.

2. Affichage conditionnel :

Des petits bugs visuels ont été réglés dans la foulée, le bouton de favoris était montré constamment même lorsqu'aucune ville n'était affichée. Il n'était donc pas utile d'afficher ce bouton et il a été retiré s'il n'y a pas de météo d'une ville affichée. De même pour le descriptif de l'écran de recherche qui se superposait parfois avec la carte de météo principale.

3. Loader :

La dernière feature sur l'affichage qui a été implémentée est le loader. En effet, il n'y avait aucune information visuelle lorsque l'application cherchait la météo de la ville demandée dans l'écran de recherche ou de préférences. Un simple loader circulaire tournant à été ajouté et un service pour les tâches UI à été créé pour n'avoir à montrer et cacher le loader qu'avec 2 appels de fonction.