UNIVERSITÉ SORBONNE PARIS NORD

IUT Villetaneuse
Université Sorbonne Paris Nord

Sélecteur multimodal : du texte à l'image

Équipe Aura+5

Orbiscope App

Janvier 2025

Réalisé par : Lydia AKMOUSSI, Nour Elhouda BOUAMLAT, Papa Bassirou DIOP, Manon GLAUDE, Lina MEDANI et Hiba OUMSID.

Table des matières

Introduction

L'application Orbiscope a été réalisée par l'équipe Aura+5 dans le cadre d'une Situation d'Apprentissage et d'Évaluation, dite SAÉ, lors du semestre 5 de la formation de BUT informatique, parcours Réalisation d'applications, à l'IUT de Villetaneuse. Le projet a débuté en septembre 2024, par la réalisation d'une vidéo de sensibilisation à la cybersécurité. Puis, il a été question d'organiser le projet et l'équipe afin de créer ce qu'on appelle un Proof Of Concept (POC), ainsi qu'une version finalisée en janvier 2025. L'équipe Aura+5 est constitué de 6 étudiants : Lydia AKMOUSSI, Nour Elhouda BOUAMLAT, Papa Bassirou DIOP, Manon GLAUDE, Lina MEDANI et Hiba OUMSID.

Les objectifs de ce projet sont de mettre en pratique nos acquis lors de notre parcours d'études, de développer un sélecteur multimodal d'images à partir de requêtes textuelles asynchrones, et de fournir une application développée en équipe, optimisée, peu gourmande en ressource, et avec un aspect commercial travaillé.

Il est donc légitime de se demander comment nous avons développé une application optimisée de sélection multimodale de drapeaux nationaux en la rendant attrayante d'un point de vue commercial.

Pour répondre à cette interrogation, nous vous présenterons dans un premier temps l'organisation de notre travail d'équipe. Dans un second temps, nous présenterons l'application du point de vue de son ingénierie logicielle. Ensuite, nous vous montrerons comment l'application est optimisée, avant d'aborder les aspects communicationnels du projet.

Section 1 - Organisation du travail d'équipe

A - Communication interne

La communication s'est effectuée à la fois en direct - lorsque nous nous retrouvions à l'IUT pour suivre des cours - mais également par l'intermédiaire d'un serveur Discord. Ce serveur, réservé à cette SAÉ, est organisé en plusieurs canaux de discussion, appelés "salons". En plus du salon principal appelé "général", un salon spécifique est dédié à la vidéo de sensibilisation à la cybersécurité. Le deuxième salon permet de planifier les réunions. Le troisième salon est lié aux activités du projet gitlab. Un dernier salon est consacré aux documents et liens utiles et importants du projet.
Les réunions d'équipe se sont tenues à l'IUT, ou bien à distance par appel sur le serveur. Nous fixions la date, l'heure et le lieu des réunions à l'avance.

B - Outils collaboratifs

Ensuite, afin de favoriser la communication interne, nous avons mis en place des outils collaboratifs. En effet, pour la gestion des documents, nous avons créé un compte Google commun et un Drive. Cela nous a permis de manipuler des fichiers partagés "Docs" et "Sheets", ainsi que des documents pdf, et des vidéos. De plus, un projet Figma a été mis en place pour que chacun puisse consulter et modifier les maquettes de l'application. En ce qui concerne le développement concret de l'application, nous avons créé un dépôt partagé sur le Gitlab de l'Université Sorbonne Paris Nord, ainsi qu'une base de données accessible à tous les membres de l'équipe, sur le site de MongoDB.

C - Attribution des rôles

Pour terminer, nous nous sommes adaptés aux appétences de chacun afin d'attribuer des rôles au sein de l'équipe, et faciliter la répartition des tâches. Chaque membre de l'équipe a eu un rôle principal, mais a pu contribuer secondairement à d'autres parties du projet. Ainsi, Bachir est le développeur back-end de l'équipe. Lydia est la responsable des tests. Nour et Lina s'occupent principalement du côté visuel du projet, à savoir le front-end, le logo, et le branding, tout en contribuant également à certaines parties du backend. Hiba est développeuse full-stack. Manon, qui a été désignée comme la "cheffe d'équipe", est chargée de coordonner l'équipe, de s'assurer de l'avancement et de la cohérence des tâches effectuées, et de gérer les réunions ainsi que le projet.

Section 2 - Ingénierie logicielle

A - Présentation de l'application

A.1 - Fonctionnement de l'application

Notre équipe est une adepte des pays. Nous aimons voyager et découvrir le monde. L'un d'entre nous connaît les capitales et drapeaux de quasiment tous les pays du monde. Ainsi, dans l'objectif d'accéder rapidement aux informations de n'importe quel pays en utilisant les langues les plus répandues dans le monde, nous avons choisi de créer un sélecteur de pays.

Nous proposons deux versions de cette application: une version **Desktop** et une version **mobile**, qu'on va vous présenter successivement ci-dessous.

A.1.a- Application mobile :

Cette version de notre application présente sélecteur multimodal de pays, qui permet de rechercher des pays en fonction de requêtes textuelles et vocales.

En entrée, on peut insérer - en anglais, en français ou en espagnol - le nom, le code, la capitale, le continent, les couleurs du drapeau, ou encore les éléments descriptifs du drapeau d'un pays. Nous pouvons également filtrer la catégorie de notre requête textuelle en cliquant sur l'un des six boutons "Nom", "Capitale", "Continent", "Code", "Couleur" et "Symbole". Il est important de noter que par défaut, la requête de l'utilisateur est considérée comme un nom de pays.
En sortie, une liste de pays correspondant à la recherche apparaît. Les résultats sont retournés de façon asynchrone, sous forme de drapeaux associés au nom de leur pays, et à leur population.
L'utilisateur a la possibilité de cliquer sur un pays, et d'accéder ainsi à une page proposant plusieurs informations sur le pays en question. Les informations fournies sont le nom officiel, la capitale, la population, les langues parlées, les monnaies, les fuseaux horaires, et le blason du pays. De plus, nous pouvons accéder à la position du pays sur Google Maps et OpenStreetMap.

A.1.b- Application Desktop:

Cette version de notre application présente sélecteur multimodal de pays, qui permet de rechercher des pays en fonction de requêtes textuelles.

En entrée, on peut insérer - en anglais, en français- le nom, le code et la capitale ainsi qu'en anglais uniquement le continent, les couleurs du drapeau, ou encore les éléments descriptifs du drapeau d'un pays. Nous pouvons également filtrer la catégorie de notre requête textuelle en cliquant sur l'un des six boutons "Nom", "Capitale", "Continent", "Code", "Couleur" et "Symbole".

Il est important de noter que par défaut, la requête de l'utilisateur est considérée comme un nom de pays.

En sortie, une liste de pays correspondant à la recherche apparaît. Les résultats sont retournés de façon asynchrone, sous forme de drapeaux associés au nom de leur pays.

L'utilisateur a la possibilité de cliquer sur un pays, et d'accéder ainsi à une page proposant plusieurs informations sur le pays en question. Les informations fournies sont le nom officiel, la capitale, la population, les langues parlées et les monnaies.

A.2 - Les versions de l'application

Il existe plusieurs versions de notre application.

Tout d'abord, pour réussir à effectuer les fonctionnalités de base, nous avons réalisé un proof of concept, qui prend la forme d'un site web. Dans cette version, nous avions uniquement la possibilité de rechercher des pays sans utiliser de filtres de catégorie. Nous ne pouvions pas accéder à une page d'information en cliquant sur un pays.

Puis, une fois que le premier poc a été terminé, nous en avons réalisé un deuxième dont l'objectif était d'apprendre à passer de l'application web à une application de bureau en apprivoisant un nouveau framework prévu à cet usage.

Notre outil est disponible sous forme d'application mobile, et d'application de bureau. C'est à cette étape que nous avons mis en place la page d'information d'un pays, ainsi que la recherche par catégorie (capitale, code, couleur, etc.), mais également que l'accès àc Google Maps et à OpenStreetMap pour l'application mobile uniquement.

Il est utile de noter que l'application mobile est actuellement fonctionnelle, tandis que l'application de bureau est toujours en cours de développement. Bien sûr, le code est accessible sur le dépôt gitlab du projet.

Nous avons choisi d'effectuer le proof of concept comme site web, car c'est le format auquel nous sommes le plus habitués, et cela nous a permis de nous concentrer davantage sur le défi technique du projet, c'est-à-dire la mise en place de la base de données et la création du moteur de recherche. Une fois que nous avons compris l'aspect technique, nous avons voulu nous lancer le défi de créer une application. En effet, le format d'application mobile et de bureau est bien plus adapté et intéressant pour ce projet.

A.3 - Structure de l'application

A.3.1 - Technologies utilisées

D'abord, pour réaliser le proof of concept, nous avons utilisé plusieurs technologies. Le développement s'effectue sur l'IDE Visual Studio Code. Coté frontend, nous avons utilisé

HTML, CSS et JavaScript, et concernant le backend nous avons utilisé Node.js avec Express. De plus, pour la version bureau du poc, nous avons intégré le framework Electron.js. Et nous avons choisi MongoDB comme système de gestion de base de données.
Ensuite, nous avons utilisé Android Studio comme IDE pour le développement de l'application mobile. Les technologies employées incluent JavaScript et XML pour la structure et le style, ainsi que les frameworks Express et React Native pour la logique applicative. Le code est quant à lui écrit en Java/Kotlin.
En ce qui concerne la version desktop de l'application, nous avons utilisé Electron.js associé à Visual Studio Code comme environnement de développement. Nous avons également repris les technologies employées lors du proof of concept, comme HTML, CSS, JavaScript ainsi que Node.js et Express pour le backend.
Le projet se divise en deux principales composantes : le côté client (frontend) et le côté serveur (backend).

A.3.2 - Frontend - côté client

D'une part, le frontend de l'application, quelle que soit sa version, repose sur plusieurs fichiers essentiels qui assurent son bon fonctionnement. D'abord, la gestion de la recherche dynamique des pays est assurée par le fichier *"front_poc/script.js"* dans le proof of concept (POC), tandis que pour l'application mobile, cette fonctionnalité est prise en charge par le fichier *"front-end/AppMobile/App.js",* et pour l'application Desktop, c'est le fichier "front_app/acceui.js" qui s'en charge. Ensuite, le contenu textuel de l'application est défini dans le fichier *"front_poc/index.html"* pour le POC et dans *"front-end/AppMobile/App.js"* pour l'application mobile, et dans "front_app/accueil.html" pour l'application desktop. Enfin, l'aspect visuel et le design de l'application sont gérés par le fichier *"front_poc/styles.css"* dans le POC et par *"front-end/AppMobile/styles.js"* dans la version mobile, et dans "front_app/acc.css" pour l'application desktop.

A.3.3 - Backend - côté serveur

D'autre part, le backend de l'application est structuré en plusieurs répertoires et fichiers essentiels, chacun ayant un rôle clé pour le bon déroulement de notre application.

Tout d'abord, le répertoire "Models" contient la définition du modèle de données de l'application, ce qui permet de structurer et de gérer efficacement les informations stockées dans la base de données.

Ensuite, le répertoire "Controllers" regroupe les fonctions fondamentales de l'application. Il est chargé de l'importation des pays à partir d'un fichier JSON, ainsi que de la recherche de pays en fonction de critères divers. Il garantit une gestion fluide des données.

Le répertoire "Data", pour sa part, contient les données provenant de l'API RestCountries sur laquelle l'application s'appuie, enrichissant ainsi les informations disponibles pour l'utilisateur.

Par ailleurs, le répertoire “Routes” établit le lien entre les requêtes des utilisateurs et la base de données. Il exploite les fonctions définies dans “Controllers” pour traiter ces requêtes et renvoyer les résultats appropriés.

De plus, le répertoire “Utils” contient des outils complémentaires, dont un fichier dédié à la traduction de divers éléments tels que les couleurs, les continents ou les symboles, facilitant l’internationalisation de l’application.

En parallèle, le fichier “enrichFlags.js” ajoute des informations détaillées sur les drapeaux. Ce qui permet de les retrouver selon leurs couleurs ou leurs symboles.

Enfin, le fichier “server.js” configure et démarre le serveur Express, gère la connexion à la base de données MongoDB et définit les différentes routes de l'API.

Concernant la base de données, l’application repose sur un schéma de données, représenté par le diagramme de la figure 1 en annexe.

A.4 - Architecture de l’application

A.4.1 - Justification de nos choix technologiques

Premièrement, nous avons choisi React.js pour son utilisation de JavaScript, que l’équipe maîtrisait déjà grâce au POC, facilitant ainsi la transition vers le développement mobile. React.js est un framework populaire pour les applications mobiles, apprécié pour sa prise en main rapide et sa capacité à créer des applications multiplateformes (iOS et Android).

Deuxièmement, le choix de Node.js s’est fait en raison de la familiarité de l’équipe avec cette technologie, ce qui a accéléré le développement. Il utilise le format JSON, cohérent avec l'API choisie, et est parfaitement compatible avec MongoDB, simplifiant la gestion des données.

Troisièmement, nous avons opté pour MongoDB en raison de l’expérience préalable de l’équipe avec cette base de données et de son aptitude à gérer des données déstructurées. Sa compatibilité avec Node.js et Express.js a assuré une intégration fluide dans notre architecture.

Par ailleurs, pour le backend, Express.js a été choisi pour son intégration naturelle avec Node.js et sa compatibilité avec MongoDB, facilitant la gestion des requêtes HTTP.

Enfin, pour l’application desktop, Electron.js a été sélectionné pour sa capacité à créer des applications multiplateformes avec les mêmes technologies que pour le frontend, optimisant ainsi le processus de développement.

A.4.2 - Architecture logicielle choisie

Nous avons décidé que l’application suivrait une architecture à trois tiers (frontend, backend et base de données), étant donné qu’il s’agit d’une application avec une unique fonctionnalité principale mais complexe. Cette architecture assure une séparation distinguable des

responsabilités, ainsi qu'une lisibilité du code sans égal. De plus, cette approche facilite la répartition des tâches au sein de l'équipe, et l'évolution indépendante des couches. À chaque version de l'application, nous maintenons cette distinction entre le backend et le frontend.

B - Lien vers le gitlab de l'application

Voici le lien vers le Gitlab de de notre application :
https://gitlab.sorbonne-paris-nord.fr/12314691/aura-plus-5

C - Lien vers la vidéo de présentation de l'application

Voici le lien vers la vidéo de présentation de l'application :
video_app_mobile_voix_off.mp4

Section 3 - Optimisation

A - Algorithme de sélection

Dans notre application, nous avons optimisé l'algorithme de sélection grâce à MongoDB pour interroger rapidement les données

- **Filtrage intelligent** : Les données sont classées en deux types, système et API, pour des requêtes plus précises.
- **Tri efficace** : Seules les 100 entrées les plus récentes sont traitées, triées par date, pour garantir rapidité et pertinence.
- **Analyse ciblée** : Les statistiques API peuvent être filtrées par endpoint (ex.: /api/countries) pour extraire des informations précises.

Ces optimisations rendent notre système rapide, pertinent et adapté à de gros volumes de données.

B - Métrique de rapidité des réponses

La rapidité de réponse des API est une métrique essentielle pour évaluer les performances de notre application backend. Pour cette analyse, nous collectons les données suivantes :

- **Temps minimum et maximum** : Identifiés grâce à `Math.min()` et `Math.max()`.
- **Temps moyen** : Calculé en divisant la somme des temps de réponse par le nombre total de requêtes.
- **Affichage clair** : Les résultats sont arrondis à deux décimales (`.toFixed(2)`) pour simplifier l'analyse.

Grâce à cette métrique, nous pouvons rapidement identifier les endpoints ayant des performances sous-optimales et diagnostiquer les problèmes potentiels.

C - Complexité et formalisation de notre algorithme de recherche et d'accès aux images à la volée

L'accès aux données est principalement basé sur MongoDB, et sa complexité dépend de la structure des requêtes :

1. **Requêtes simples (type ou endpoint)** :
    - Complexité moyenne : **O(n)**, où *n* est le nombre d'enregistrements correspondant au filtre. MongoDB applique des filtres efficacement, mais si un index est utilisé, la complexité peut être réduite à **O(log n)**.

2. **Tri des résultats** :
    - MongoDB trie les données sur le serveur avant de les envoyer. Si les données triées sont indexées, le tri est optimisé, ce qui minimise l'impact sur les performances.
3. **Calcul des statistiques** :
    - Une fois les données extraites, les statistiques des temps de réponse (min, max, moyenne) sont calculées côté serveur avec une complexité **O(n)** pour traverser la liste des résultats.

D - Description de notre approche et implémentation (Graphes)

D.1 - Contexte et choix techniques

Initialement, nous avions prévu d'utiliser Grafana pour la visualisation des métriques système et API. Cependant, des complications liées à l'URL de connexion nous ont poussés à adopter une approche différente. Nous avons donc implémenté une solution front-end personnalisée avec **React Native**, en développant un composant appelé **MetricDashboard.js**. Ce fichier permet de récupérer, d'analyser et de visualiser les données collectées par notre backend.

D.2 - Fonctionnement de `MetricDashboard.js`

Le fichier **`MetricDashboard.js`** est un composant React Native qui remplit plusieurs rôles :

1. **Récupération des données depuis le backend** :
    - Nous effectuons deux requêtes HTTP via `fetch()` vers les endpoints suivants :
        - `/metrics/api/stats` : Fournit des statistiques détaillées sur les temps de réponse des API (temps minimum, moyen et maximum).
        - `/metrics/system` : Fournit les 100 dernières métriques système, incluant l'utilisation du CPU, de la mémoire, et le temps de fonctionnement.
2. **Traitement des données** :
    - Une fois les données récupérées, nous les transformons en un format adapté à la visualisation avec la bibliothèque `react-native-chart-kit`.
3. **Visualisation** :
    - Nous utilisons deux types de graphiques :
        - **Graphique à barres** (BarChart) pour afficher les statistiques API : temps minimum, moyen et maximum.
        - **Graphique linéaire** (LineChart) pour afficher les tendances des métriques système au fil du temps.

D.3 - Liens avec le backend et explication des processus

1. Collecte et enregistrement des données

Dans le backend, les données sont collectées et enregistrées à travers deux mécanismes principaux :

- **Métriques API** : Les statistiques sur les temps de réponse sont calculées dynamiquement à partir des données stockées dans MongoDB, grâce au contrôleur `getApiResponseStats`. Ce contrôleur filtre les métriques par endpoint, calcule les statistiques (min, moyenne, max) et les renvoie au frontend.
- **Métriques système** : Les métriques telles que l'utilisation du CPU, de la mémoire, et le temps de fonctionnement sont collectées toutes les 5 minutes par un cron job. Ces données sont ensuite stockées dans MongoDB via le modèle `MetricsModel`.

2. Transmission des données au frontend

- Les données sont récupérées en temps réel via les endpoints `/metrics/api/stats` et `/metrics/system`.
- Le frontend traite ces données pour les représenter visuellement et permettre une analyse immédiate.

D.4 - Analyse des tendances et impact sur le système

1. **Évolution des métriques API** :
    - Si des pics de temps de réponse sont détectés, cela peut indiquer une surcharge temporaire du serveur ou un problème spécifique avec un endpoint.
    - Nous pourrions configurer des alertes dans le futur pour notifier l'équipe en cas de dépassement d'un seuil critique (par exemple, un temps de réponse supérieur à 1 seconde).
2. **Tendances des métriques système** :
    - Une augmentation constante de l'utilisation du CPU ou de la mémoire peut signaler un problème sous-jacent, comme des processus mal optimisés ou des fuites de mémoire.
    - Ici, les tendances sont globalement stables, ce qui indique que le système est bien dimensionné pour la charge actuelle.

Par exemple en tant réel les images suivantes montrent les graphes qu'on a pu générer depuis notre front-end React-Native

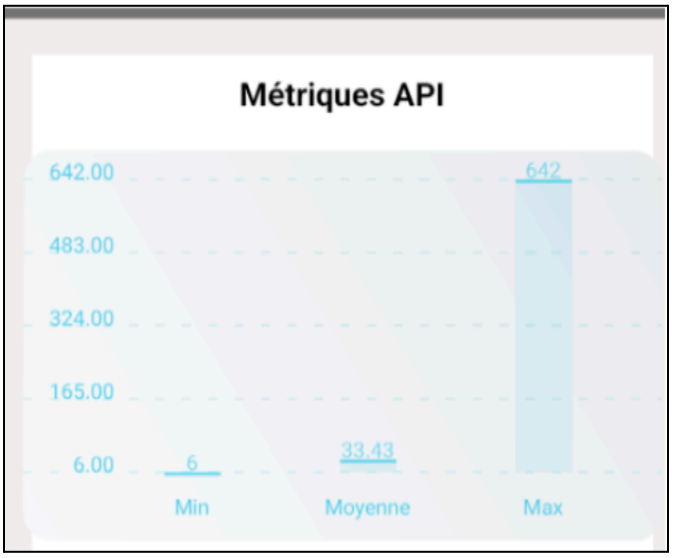


Figure 2 : graphique des métriques API

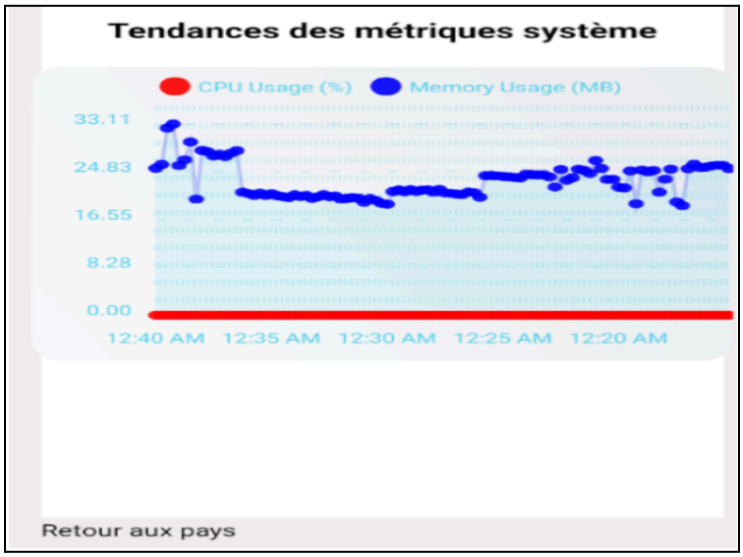


Figure 3 : graphiques des tendances des métriques système

D.5 - Analyse du graphique des métriques API

Le graphique des métriques API présente trois barres :

- **Min (6 ms)** : Indique des requêtes rapides et optimisées.
- **Moyenne (33,39 ms)** : Montre une performance stable pour la majorité des requêtes.
- **Max (642 ms)** : Reflète des cas plus longs dus à la complexité des requêtes, une surcharge serveur, ou un volume important de données.

Implications :

- Une valeur maximale élevée peut pointer des endpoints nécessitant des optimisations (requêtes coûteuses, jointures multiples, etc.).
- Les barres évoluent en fonction de la charge et des optimisations effectuées.

Actions recommandées :

- Identifier les requêtes lentes via les logs et optimiser les endpoints.
- Ajouter des index à la base de données et améliorer le traitement backend.

D.6 - Analyse du graphique des tendances des métriques système

Le graphique illustre deux métriques :

- **Série rouge (CPU)** : Faible utilisation, montrant un serveur sous-utilisé et bien optimisé. Les pics observés correspondent à des tâches ponctuelles (cron jobs, requêtes complexes).
- **Série bleue (Mémoire)** : Variations normales liées à la gestion de mémoire par Node.js, sans fuite mémoire apparente.

Implications :

- Une faible utilisation CPU et des fluctuations maîtrisées de mémoire indiquent un système stable.
- Une augmentation constante en bleu ou des pics rouges répétés pourraient signaler des problèmes à surveiller.

Pourquoi c'est utile :

Ces graphiques permettent une surveillance en temps réel, aident à détecter les anomalies (temps de réponse élevés, surconsommation mémoire) et guident l'optimisation continue.

E - Validation fonctionnelle et tests réalisés

- **Tests de performance :** Les tests de performance ont été réalisés dans le cadre du POC afin d'évaluer la capacité de l'application à répondre rapidement et efficacement aux requêtes des utilisateurs, même sous des charges importantes.

1. **Objectifs des tests :**
   Évaluer la rapidité et la stabilité des routes API critiques sous différentes charges simulées.
   Identifier les limitations dans les temps de réponse et la gestion des utilisateurs simultanés.
   Valider la robustesse des fonctionnalités essentielles, telles que l'importation de données et la recherche.

2. **Méthodologie :**
   Outil utilisé : Artillery — un outil spécialisé dans les tests de charge et la génération de scénarios de performance.

   Scénarios testés :

   - Importation massive de données via la route `/countries/import`.
   - Recherches simples et avancées sur `/countries`.
   - Vérification des erreurs sur les routes invalides (`404`).

   Configuration de l'environnement :
   Les tests ont été exécutés sur une base de données MongoDB Atlas, utilisée à la fois pour le développement et les tests.

3. **Résultats obtenus**
   - **Charge testée :** 300 utilisateurs simultanés.
   - **Temps de réponse moyen :** 1,1 seconde.
   - **Taux de réussite :** 100 % des requêtes ont obtenu des réponses valides.
   - **Erreurs simulées :** Les routes non valides ont correctement retourné des erreurs `404`.

   Résumé des métriques clés Artillery :

   - **Temps de réponse maximum :** 10 secondes (pendant des pics de charge).
   - **Temps de réponse moyen pour 95 % des requêtes :** 2 secondes.
   - **Temps de réponse minimum :** < 1 seconde (réponses optimales).

- **Tests d'intégration :** Les tests d'intégrations ont été réalisés pour valider les intéractions entre les composants principaux du backend.

1. **Objectifs des tests**

❖ Vérifier la bonne gestion des routes principales, notamment :
  ➢ **Importation des données JSON** via `/countries/import`.
  ➢ **Recherche de pays** via `/countries/search`.
  ➢ Gestion des routes non valides avec un retour d'erreur 404.

❖ Assurer une intégration fluide entre le backend Express, la base de données MongoDB et les données JSON.

❖ Valider la logique métier pour le formatage et la manipulation des données importées.

2. **Outils utilisés :**

Jest : Framework de tests pour exécuter et valider les tests.
Supertest : Utilisé pour simuler des requêtes HTTP vers les routes backend**.**

3. **Scénarios téstés :**

- **Importation de pays :** Validation de l'importation des données JSON dans MongoDB et gestion des erreurs lorsque le fichier JSON est introuvable.

- **Recherche de pays :** Vérification de la recherche par mot-clé (nom de pays ou traduction) et validation de la pertinence des résultats obtenus.
- **Routes inexistantes :** Vérification que les routes non définies retournent une erreur 404.

4. **Résultats obtenus :**

Les tests d'intégration ont confirmé que les données JSON sont correctement importées dans la base MongoDB, avec suppression préalable des anciennes données. Les recherches par nom ou traduction fonctionnent comme attendu, retournant des résultats pertinents et un statut 200 OK. Enfin, les erreurs, comme les routes non valides, sont bien gérées et renvoient une réponse 404 avec un message clair.

5. **Limitations :**

- Utilisation d'une base MongoDB partagée pour les tests et le développement, ce qui peut entraîner des conflits de données.
- Absence de tests pour certaines fonctionnalités spécifiques à l'application mobile.
- Les tests ne couvrent pas encore tous les scénarios possibles, notamment les cas d'erreurs avancés.

Section 4 - Aspects communicationnels

A - Charte graphique de l'application

Des captures d'écran de l'application mobile se trouvent en annexe.

A.1 - La palette de couleurs

La palette de couleurs a été pensée pour rendre l'application claire et lisible. Les drapeaux étant déjà très colorés, nous avons opté pour des couleurs simples et sobres afin de les mettre en avant, d'éviter la surcharge visuelle et d'assurer une esthétique cohérente.

Le fond, en blanc cassé ou gris, a été choisi pour rester discret. La barre de recherche, les boutons de catégories et les cadres des résultats sont en blanc, une couleur qui attire l'œil sans le fatiguer, facilitant ainsi la concentration sur les drapeaux. Le texte noir contraste avec le fond clair, garantissant une lecture fluide.

Pour la page d'informations d'un pays, la même approche est appliquée : un fond blanc et un texte noir assurant lisibilité et mise en valeur du drapeau et du blason. Deux boutons permettent d'accéder à Google Maps et OpenStreetMap, respectivement en vert et gris foncé, ce dernier étant moins visible car secondaire. Enfin, le bouton de fermeture est rouge, respectant la convention associée à cette fonction.

A.2 - Typographie

Les polices d'écritures que nous avons choisies sont simples et lisibles. Comme dit précédemment, nous primons sur la lisibilité et la clarté des informations, plutôt que de choisir une police plus fantaisiste mais moins compréhensible.

A.3 - Iconographie

Nous avons également choisi une icône que l'on retrouve fréquemment dans d'autres applications et sites web. Par exemple, la loupe qui représente la barre de recherche est un élément très connu. En effet, cela permet à l'utilisateur de se repérer facilement et rapidement dans l'application. Donc, cela agit sur l'intuitivité de notre solution.

A.4 - Mise en page

Pour la mise en page de l'application mobile, tous les éléments sont concentrés dans la zone supérieure et centrée de l'écran pour assurer leur visibilité, même avec le clavier affiché, et éviter la dispersion de l'utilisateur.

Le logo est placé en haut et au centre, affirmant l'identité de l'application. Juste en dessous, les boutons de filtres et la barre de recherche occupent toute la largeur de l'écran, avec des

marges pour une meilleure lisibilité. Les boutons sont disposés en 3 colonnes et 2 lignes pour une organisation harmonieuse.

Les résultats de recherche apparaissent sous forme de liste déroulante sous la barre de recherche, assurant une cohérence visuelle. Chaque résultat est encadré pour se démarquer du fond et offrir plus de clarté.

Enfin, les informations détaillées d'un pays s'affichent sur une page dédiée pour éviter une surcharge d'informations sur la page principale et permettre à l'utilisateur de rester concentré.

B - Logo

Notre logo est composé d'un globe terrestre à gauche, du titre de l'application à droite, et d'un fond clair. D'un côté, le goble terrestre est une sphère en 3D. Sa surface non pleine est formée par un ruban de drapeaux colorés, qui dessine ses contours. Cet élément est représentatif de notre application qui permet de faire voyager notre esprit à travers le monde avec ses drapeaux. D'un autre côté, le titre de l'application est noir et sa police est assez fantaisiste. Cela apporte de la profondeur au projet. De plus, le fond est clair et peu visible, afin de mettre en avant les autres éléments du logo.
À l'image de notre application mobile, le côté vivant et animé du logo est assuré par la variété de couleurs des drapeaux du monde. Les autres éléments sont donc consciemment gardés sobres.

C - Image de marque / Branding

C.1 - Titre de l'application

Notre application s'intitule "Orbiscope". Ce nom est composé de "Orbi" et "Scope". "Orbi", provient du terme latin "orbis", qui signifie "monde " ou "cercle". Par ailleurs, "Scope" vient du mot grec "skopos", qui a pour signification "observer", "examiner". En effet, cette application permet d'observer le monde. Nous avons choisi ces termes car ils reflètent la vision que nous avons de notre application, et rappellent sa fonctionnalité principale, qui est de trouver des informations à propos de n'importe quel pays dans le monde. De plus, ce nom évoque bien le thème principal de l'application, qui se veut universelle.

C.2 - Histoire de l'application

L'histoire de l'application Orbiscope débute à l'IUT de Villetaneuse, fin 2024. Aura+5 cherche un moyen de personnaliser son projet de fin d'étude, visant à mettre en avant ses compétences techniques et interpersonnelles.

En parallèle, les membres de l'équipe se retrouvent chaque jour dans le hall de l'université, où sont accrochées des banderoles de drapeaux nationaux. Ils se lancent alors le défi d'identifier les pays correspondants, certains vérifiant les réponses sur internet.

Progressivement, Aura+5 s'intéresse davantage aux informations sur les pays et complexifie le jeu. C'est alors que l'équipe réalise que la recherche d'informations en ligne est plus longue que prévu, nécessitant trop de sites différents. L'idée naît alors : créer un projet répondant à la fois au sujet imposé et à leur propre besoin – voire à celui d'autres personnes.

C'est ainsi qu'est née l'application Orbiscope, un sélecteur multimodal de drapeaux de pays.

C.3 - Vision et valeurs de l'application

Notre vision pour Orbiscope est de rendre accessibles les informations essentielles sur les pays du monde dans une seule application. Nous voulons mettre le monde à portée de main.

Nos valeurs reposent sur l'ouverture au monde et la culture de la différence qui nous unit. Malgré la diversité des pays, chacun appartient à un même ensemble : le monde.

C.5 - Positionnement par rapport aux concurrents

Orbiscope se démarque par son interface utilisateur simpliste et minimaliste, sa nature d'application mobile accessible sur iOS et Android (contrairement à des concurrents qui sont des sites internet), ainsi que par la gratuité de l'ensemble du produit.

D - Marketing

D.1 - Canaux de communication envisagés

Pour faire connaître notre application, nous souhaitons passer par plusieurs réseaux sociaux. En premier lieu, nous allons utiliser les plateformes Tiktok et Youtube, en mettant en place des partenariats avec des youtubeuses/youtubeurs et tiktokeuses/tiktokeurs qui publieront des vidéos dans lesquelles ils utiliseront notre application pour s'amuser, ou bien à des fins éducatives. En second lieu, nous créerons trois comptes instagram (Orbiscope_francais, Orbiscope_espanol et Orbiscope_english), sur lesquels nous publierons des photos attractives de l'application, des posts intéressants, et des vidéos de mise en situation.

D.2 - Campagnes de lancement

Le message clé que nous voulons faire passer est : "Nous pouvons accéder aux données de tous les pays en une seule application." . Nous souhaitons que la campagne de lancement adopte un ton de communication éducatif, car nous souhaitons viser des jeunes collégiens, lycéens, étudiants, et autres jeunes de moins de quarante ans, qui ont l'habitude d'utiliser leur téléphone comme source d'information, et qui sont francophones, anglophones ou hispanophones.
L'application mobile sera disponible sur les App Stores Google Play Store et Apple Store.

Conclusion

En conclusion, l'équipe Aura+5 a mené ce projet et a développé une application mobile optimisée de sélection multimodale de drapeaux nationaux en la rendant attrayante d'un point de vue commercial.

D'une part, nous avons organisé notre travail d'équipe en attribuant un rôle à chaque membre, et en mettant en place des outils de communication, comme le serveur Discord, et de collaboration, à l'image de Google Drive et de Gitlab.

D'autre part, nous avons créé un sélecteur multimodal de drapeaux nationaux complet et dynamique en choisissant une architecture trois-tiers, ainsi que des technologies adaptées telles que Node.js, React-Native et MongoDB, qui, tout comme les tests effectués, contribuent également à l'optimisation de l'application.

De surcroît, nous avons travaillé le côté communicationnel de notre application, en sélectionnant consciemment une charte graphique minimaliste pour mettre en avant les drapeaux, et pour conserver une certaine intuitivité. Nous avons également réfléchi à l'image de marque de l'application, comme son histoire, et à une approche marketing qui s'adapte à notre public cible, tel que les jeunes étudiants.

Par ailleurs, il est intéressant de songer à l'ajout de nouvelles fonctionnalités afin d'améliorer l'application. En effet, il pourrait être intéressant de pouvoir jouer l'hymne national lorsqu'on accède aux informations détaillées d'un pays, et de rendre la recherche possible dans beaucoup plus de langues courantes, telles que le mandarin et l'arabe. Aussi, nous pourrions envisager d'intégrer des fonctionnalités de fidélisation, telles que la création d'un compte, la possibilité d'ajouter des commentaires aux informations d'un pays, ou encore de personnaliser son fond d'écran.

Annexes

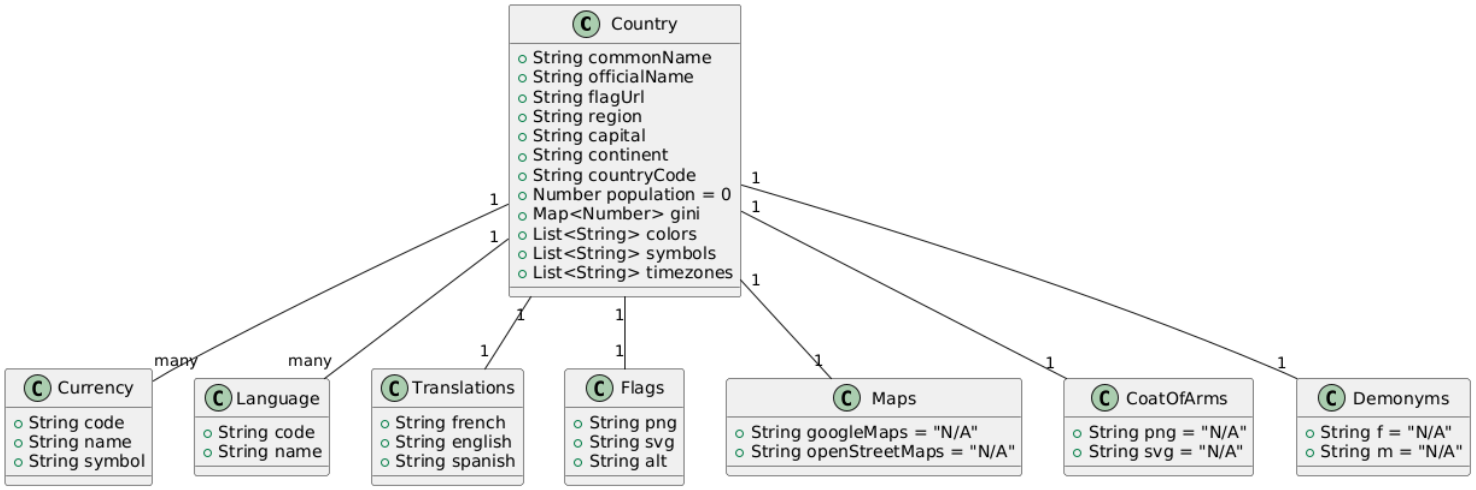


Figure 1 : diagramme représentant le modèle de données

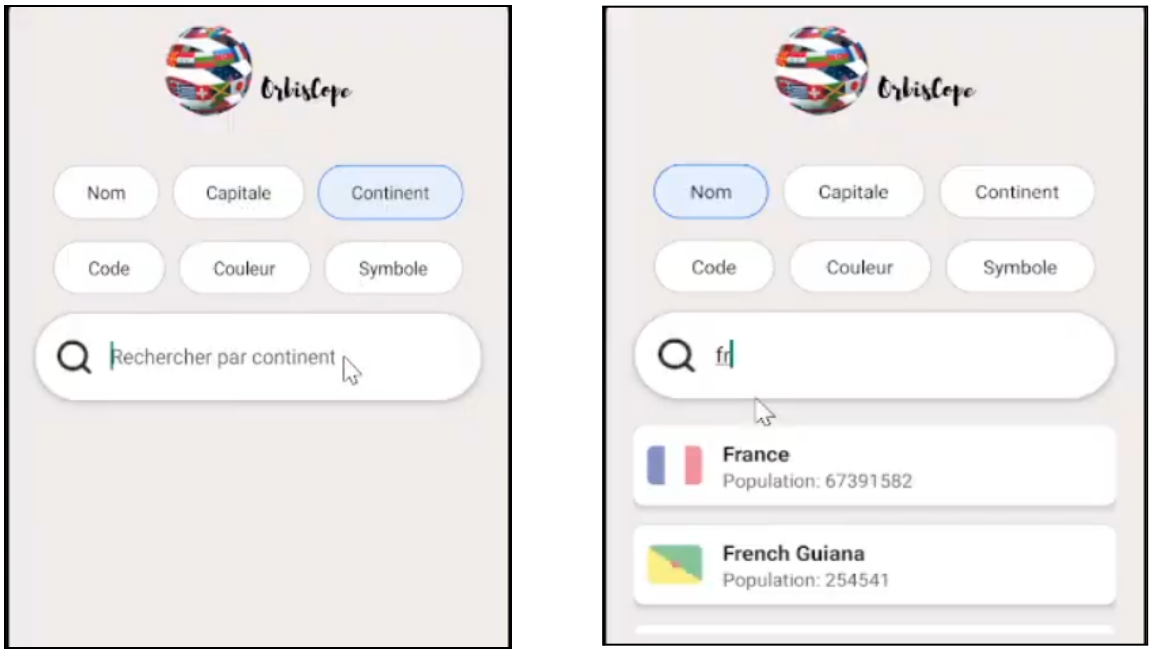


Figure 2: interface utilisateur de l'application mobile avec et sans recherches en cours

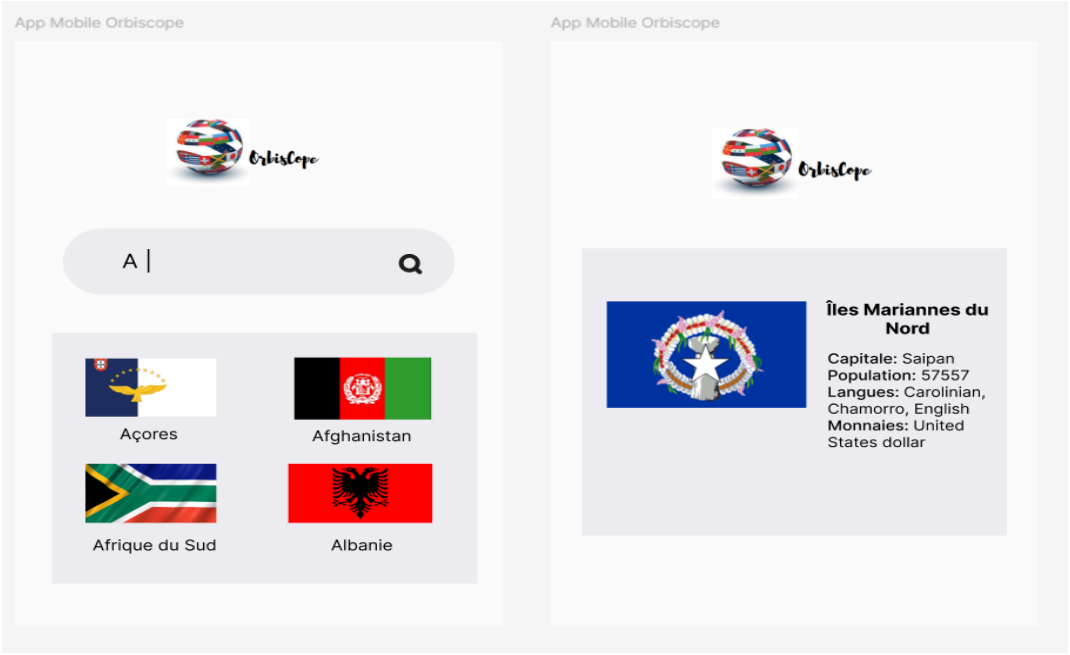
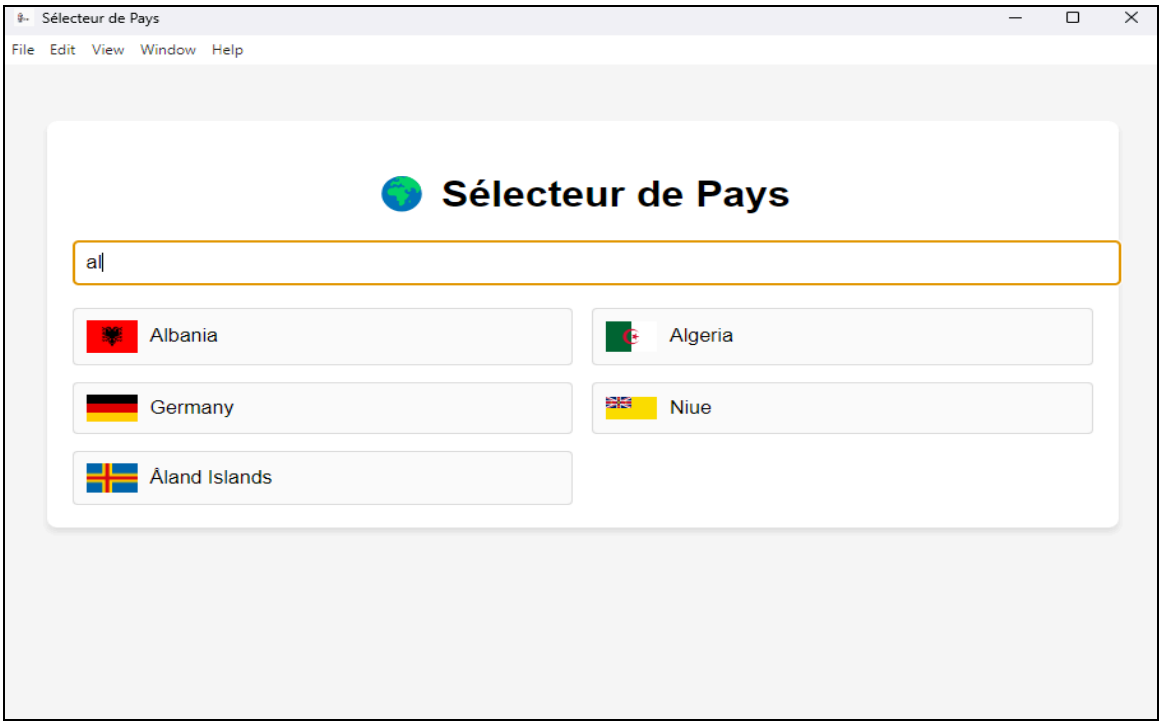


Figure 3 : Maquette de l'application mobile

Figure 4 : page d'informations sur l'Espagne après la recherche "spain"

Figure 5 : Logo de l'application



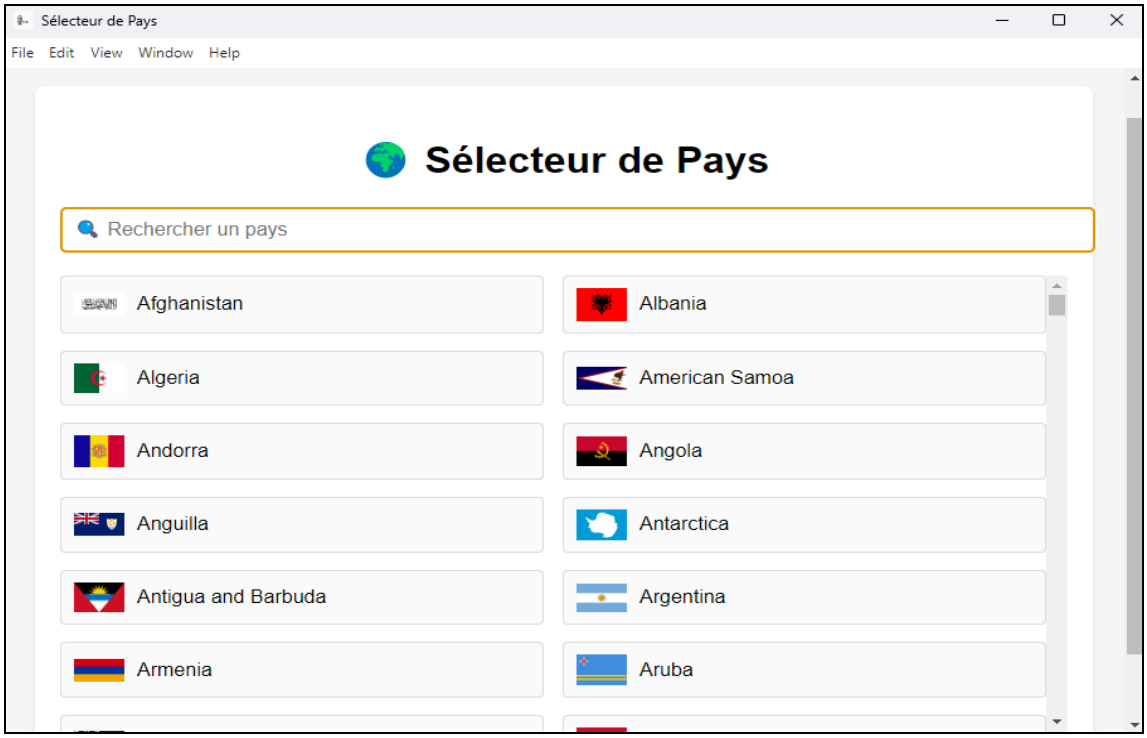


Figure 6 : Poc de l'application Desktop

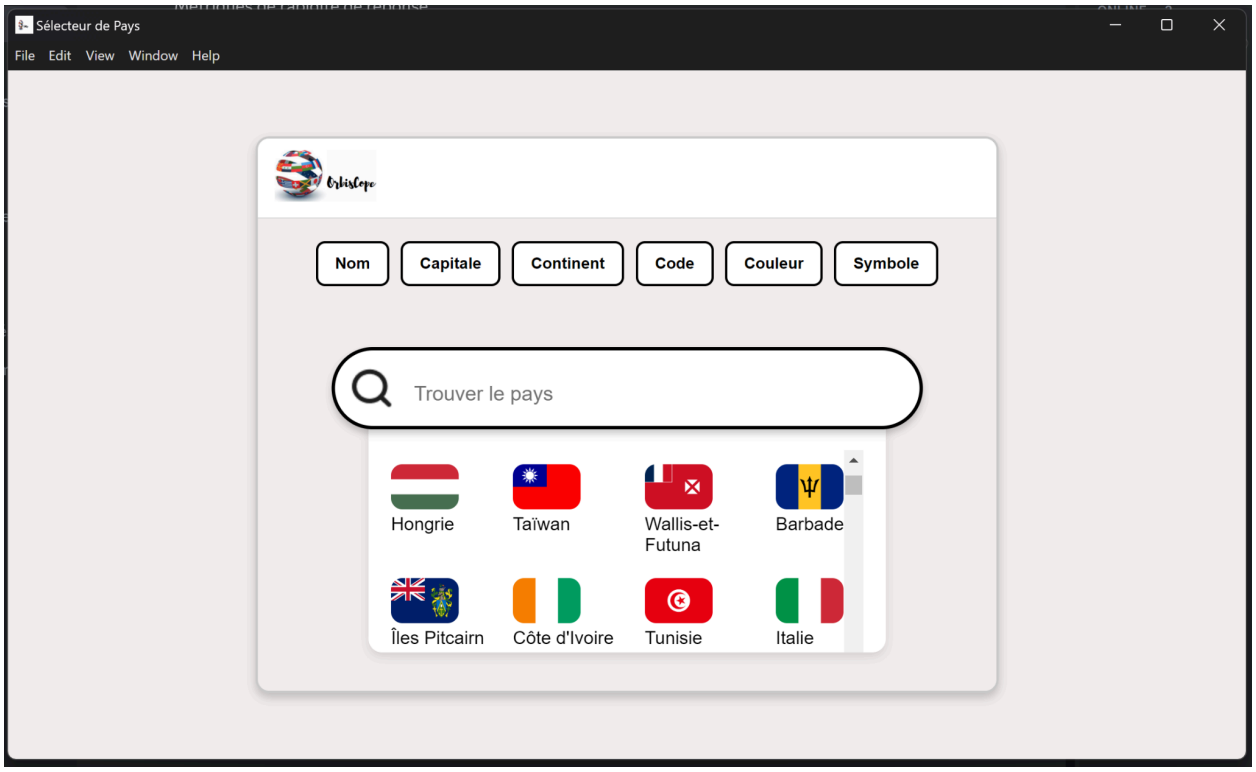


Figure 7 : Application Desktop

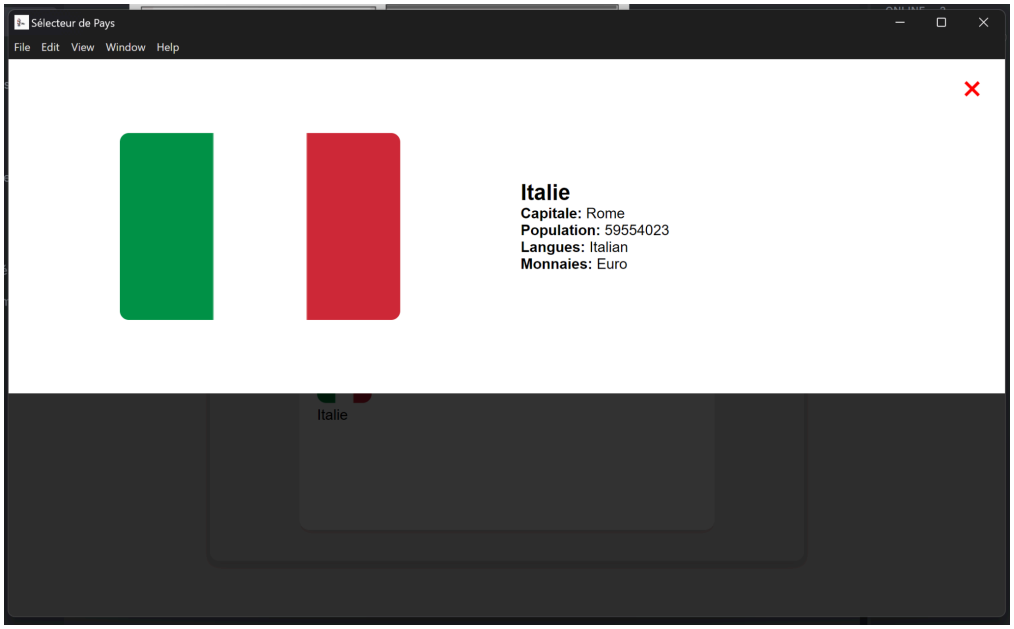


Figure 8 : Page d'information de l'italie sur l'application Desktop

Références et liens utiles

- Logo de l'Université Sorbonne Paris Nord : https://iutv.univ-paris13.fr/wp-content/uploads/LOGOTYPE-Officiel-Universite%CC%81-Sorbonne-Paris-Nord-2023.png
- Logo de l'IUT de Villetaneuse : https://iutv.univ-paris13.fr/wp-content/uploads/logotype-iutv-2023.png
- Notre projet gitlab : https://gitlab.sorbonne-paris-nord.fr/12314691/aura-plus-5
- MongoDB Atlas : MongoDB Cloud
- Notre Cahier des charges : Cahier des charges
- Nos maquettes : https://www.figma.com/design/P7xJZTwhhe1QO2cncYcJde/Untitled?node-id=0-1&p=f&t=bRgVFyrJRthHoBGU-0
- Notre vidéo de présentation : video_app_mobile_voix_off.mp4
- API RestCountries : https://restcountries.com/v3.1/all
- Notre vidéo de sensibilisation à la cybersécurité : auraplus5_lemag_sensibilisation_cybersecurite_2024.mp4
- Origine du mot "Orbi" : Correspondance pour ORBIS
- Origine du mot "Scope" : σκοπός — Wiktionnaire, le dictionnaire libre
- Générateur du diagramme UML : PlantUML Web Server