2025/2026	AP – Projet 2 – Android – Itération 2
BTS SIO 2SLAM	Auteur : PORTOLLEAU Anaïs & DELAUNAY-GUITTON Benjamin & CHARBONNEAU Tom
	Date de rédaction : 13/11/2025

Projet d'application Android de réservation pour R3st0.fr

Table des matières

2025/2026	AP – Projet 2 – Android – Itération 2
BTS SIO 2SLAM	Auteur : PORTOLLEAU Anaïs & DELAUNAY-GUITTON Benjamin & CHARBONNEAU Tom
	Date de rédaction : 13/11/2025

Itération 2 - Affichage du détail d'un restaurant

User story 1 : « En tant qu'utilisateur, je souhaite, à partir de la liste des restaurants, pouvoir consulter la fiche détaillée d'un restaurant. »

Date exigibilité : 01/12/2025 à 14H

Pour installer projet voir le README du dépôt ci-dessous (branche main)

Rappel du dépôt (aussi utilisé pour la gestion des tickets): Application Android

Lien de la branche Itération 2 (Projet Android) : iteration2_finale

Lien du dépôt GITLAB de l'API : API

Lien de la branche Itération 2 (Projet API) : iteration2

2025/2026	AP – Projet 2 – Android – Itération 2
BTS SIO 2SLAM	Auteur : PORTOLLEAU Anaïs & DELAUNAY-GUITTON Benjamin & CHARBONNEAU Tom
	Date de rédaction : 13/11/2025

Ticket 1 : Navigation entre deux écrans

Branche utilisée au moment du ticket (Application Android) : iteration2_ticket1

Programmes :

Tout d'abord, j'ai commencé par me rendre dans la classe *ListRestoActivity* dans laquelle j'ai ajouté à la ligne 97 un écouteur d'évènement sur la liste des restaurants affiché dans un ListView. A l'intérieur de celui-ci, j'ai instancié un nouvel Intent afin de naviguer de l'activity « *ListRestoActivity* » à « *DetailsRestoActivity* ».

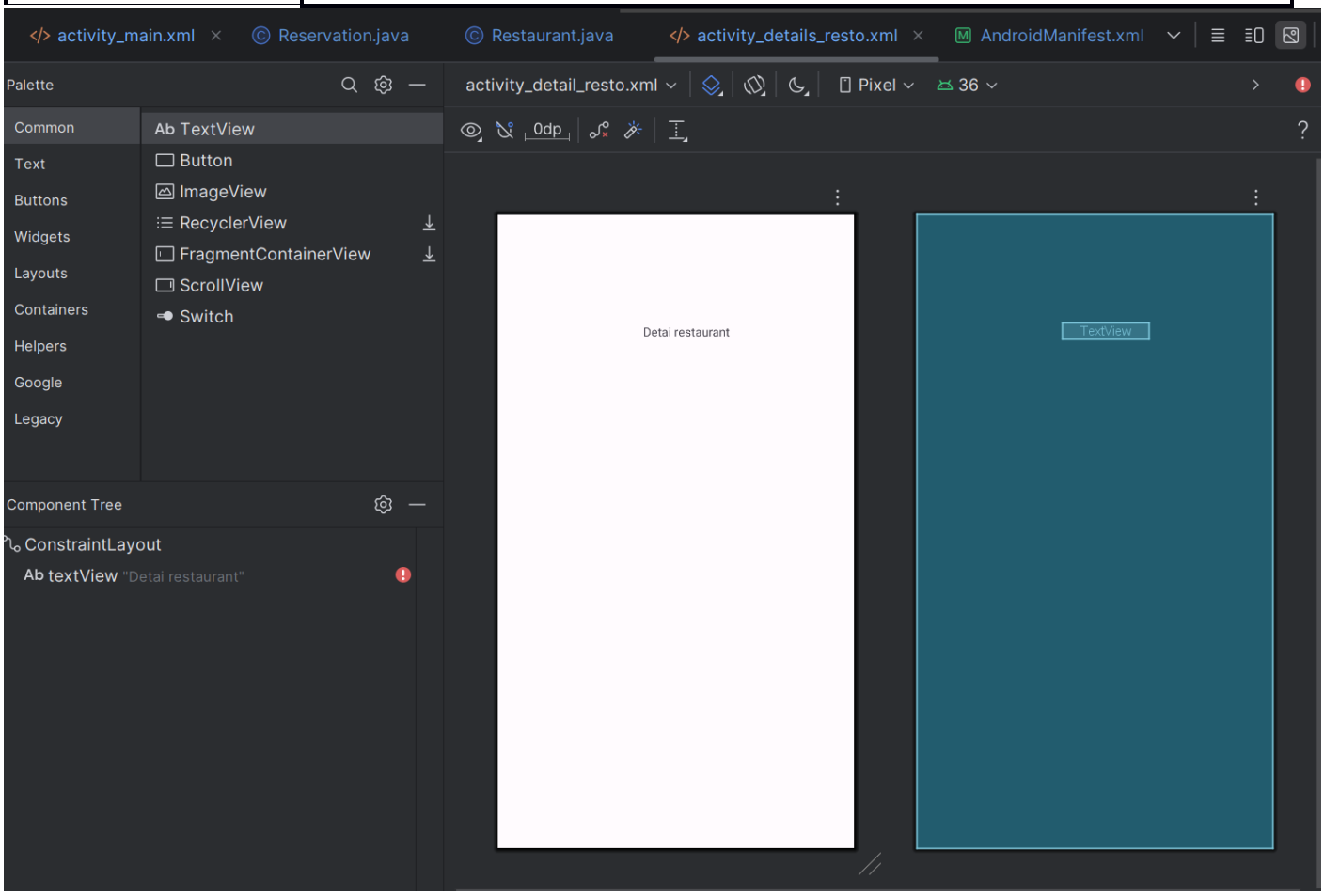
```
96        // Clic sur un élément de la liste, envoie -> sur activity_restaurant_details
97        listResto.setOnItemClickListener(( AdapterView<?> parent, View view, int position, long id) -> {
98            Restaurant resto = lesRestos.get(position);
99            Intent intent = new Intent( packageContext: ListRestoActivity.this, DetailsRestoActivity.class);
100           startActivity(intent);
101       });
```

En parallèle, j'ai créé la classe *DetailsRestoActivity* pour rendre possible la navigation entre les deux activités.

DetailsRestoActivity.java | ListRestoActivity.java | activity_details_resto.x

```
1   package com.example.p2_g7_2slam_ap_projet_android.controleur;
2
3   import ...
8
9   public class DetailsRestoActivity extends AppCompatActivity {
10      @Override
11      protected void onCreate(Bundle savedInstanceState) {
12          super.onCreate(savedInstanceState);
13          setContentView(R.layout.activity_details_resto);
14      }
15
16  }
```

Après cela, j'ai créé un nouveau layout « activity_details_resto » dans lequel on peut retrouver un composant TextView temporaire, dont l'unique but est de s'assurer que nous arrivons bien sur la bonne activité lors des tets fonctionnels.



Enfin, je n'oublie pas d'ajouter ma nouvelle activité dans le fichier manifest.

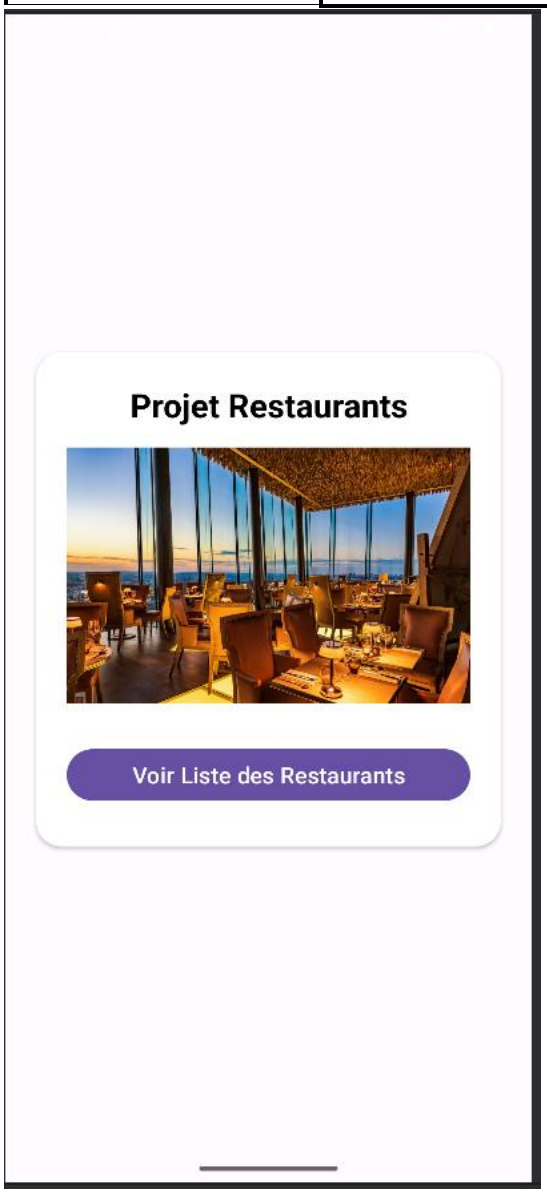
```
28        <activity android:name=".controleur.ListRestoActivity" />
29        <activity android:name=".controleur.DetailsRestoActivity" />
```

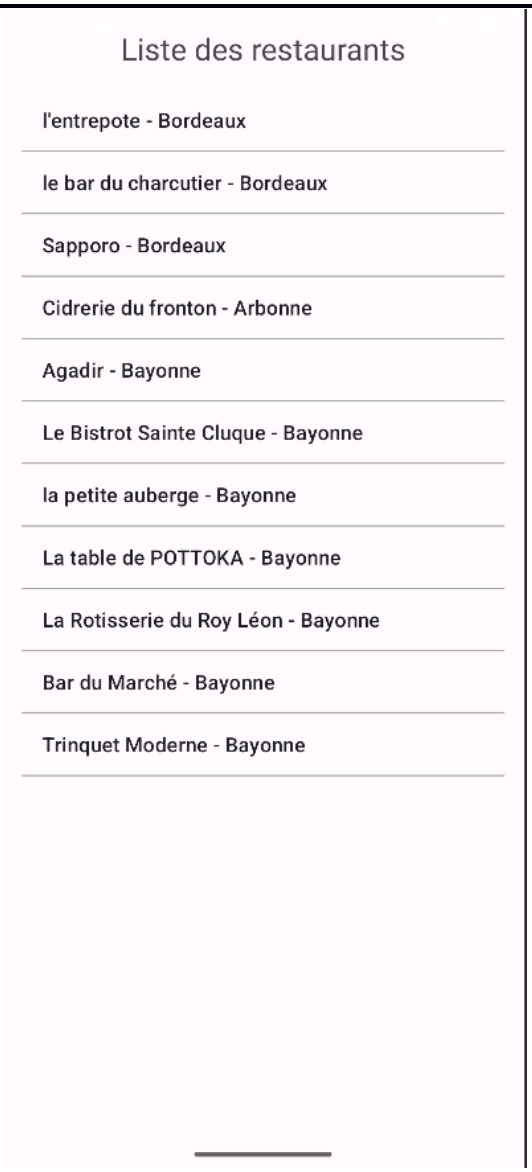
Test fonctionnel :

Dorénavant, comme nous sommes rendus à l'itération 2, nous allons partir du principe que vous savez lancer l'application. En cas de problème, référez-vous au README présent dans la branche « Main » du dépôt : Projet Android.

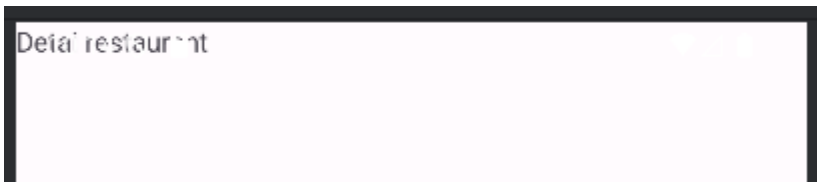
Objectif : Vérifier que l'on arrive sur le layout « activity_details_resto » lorsqu'on clique sur l'un des restaurants.

1. Une fois l'application lancée, cliquez sur le bouton « Voire Liste des Restaurants ». Puis cliquez sur l'un des restaurants.





2. Normalement, vous arrivez sur l'écran suivant qui correspond à l'activity « activity_details_resto ». On arrive à reconnaître tant bien que mal le TextView montré précédemment.



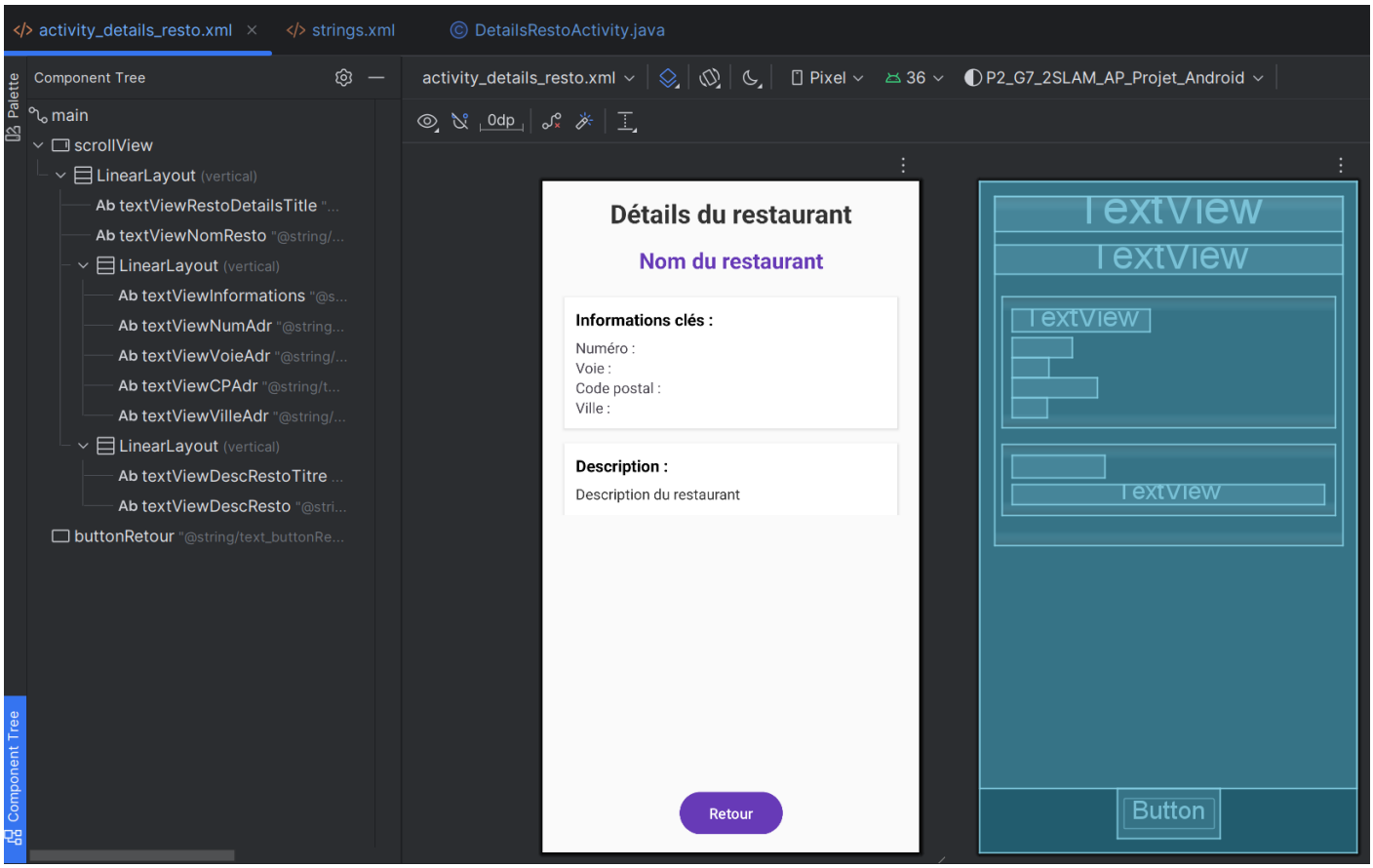
2025/2026	AP – Projet 2 – Android – Itération 2
BTS SIO 2SLAM	Auteur : PORTOLLEAU Anaïs & DELAUNAY-GUITTON Benjamin & CHARBONNEAU Tom
	Date de rédaction : 13/11/2025

Ticket 2 : UI : Détails du restaurant

Branche utilisée au moment du ticket (Application Android) : iteration2_ticket2

Programmes :

Tout d'abord, j'ai commencé par découper mon layout en 2 partie : Un scrollView et un Bouton Retour servant à revenir à la page précédente. Le scrollView contient le titre de l'écran, le nom du restaurant, ainsi que toutes les informations clés qui seront récupérées grâce à l'API.



Dans l'objectif de mettre en œuvre les bonnes pratiques de programmation, j'ai fait en sorte que les textes apparaissant sur l'application ne soit plus « brut » mais écrit dans le fichier « string.xml » qui est très utile lorsque l'on souhaite implémenter différentes langues dans notre application.

strings.xml × DetailsRestoActivity.java

Edit translations for all locales in the translations editor.

```xml
<resources>
    <string name="app_name">P2_G7_2SLAM_AP_Projet_Android</string>
    <string name="text_button_voirLaListeResto">Voir Liste des Restaurants</string>
    <string name="text_button_AffichageDetailsResto">Voir Affichage détails des Restaurants</string>
    <string name="text_button_ReservationRestos" >Voir Reservation resto</string>
    <string name="titre">Projet Restaurants</string>
    <string name="text_buttonRetour">Retour</string>
    <string name="text_codePostal">Code postal :</string>
    <string name="text_description">Description :</string>
    <string name="temp_text_description">Description du restaurant</string>
    <string name="text_informationsCles">Informations clés :</string>
    <string name="temp_text_nomRestaurant">Nom du restaurant</string>
    <string name="text_numResto">Numéro :</string>
    <string name="text_detailsResto">Détails du restaurant</string>
    <string name="text_ville">Ville :</string>
    <string name="text_voie">Voie :</string>

</resources>
```

Aussi, j'ai ajouté dans la classe *DetailsRestoActivity*, la fonctionnalité du bouton « Retour ». La méthode finish() permet dans notre cas, de revenir à la vue précédente.

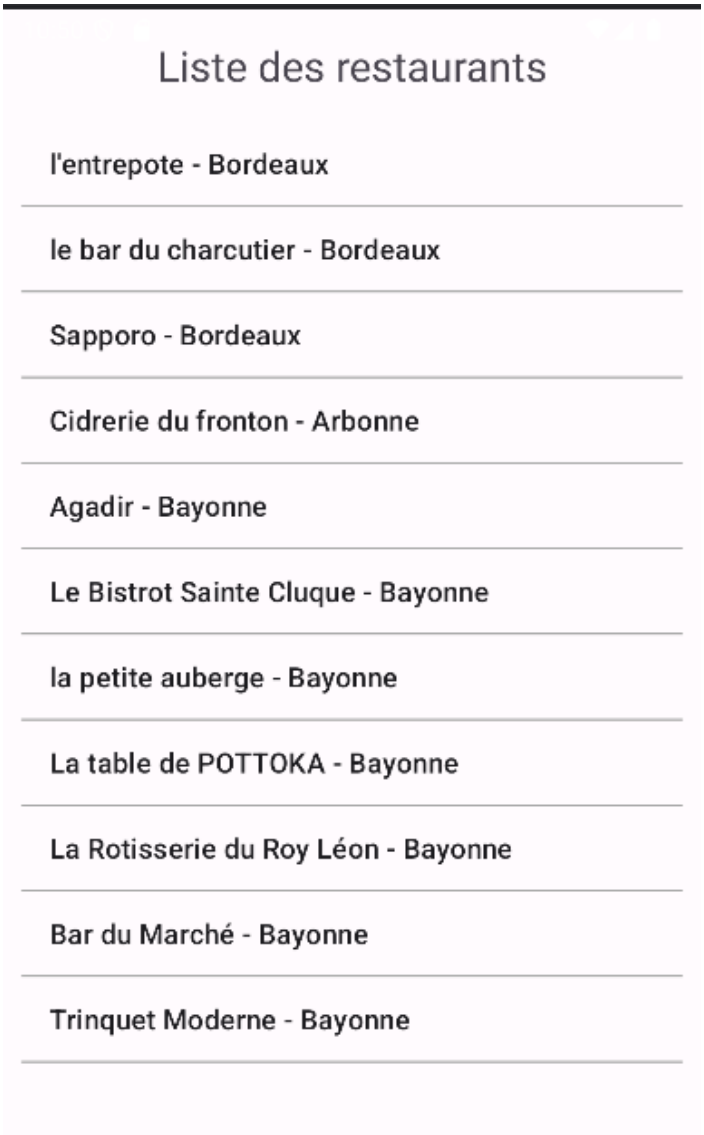
DetailsRestoActivity.java ×

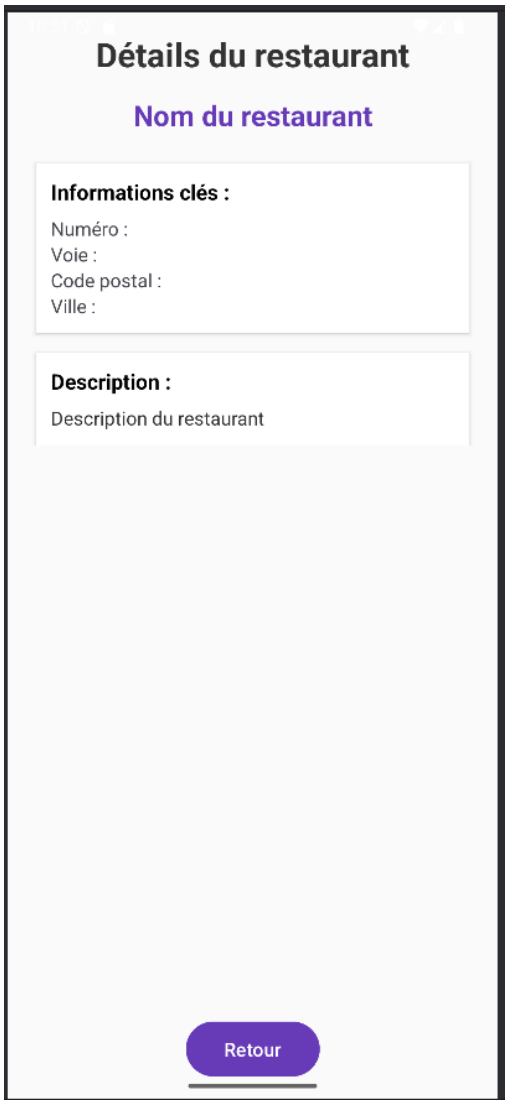
```java
package com.example.p2_g7_2slam_ap_projet_android.controleur;

import ...

public class DetailsRestoActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_details_resto);

        // Retour à l'écran précédent
        Button buttonRetour = findViewById(R.id.buttonRetour);
        buttonRetour.setOnClickListener( View v -> finish());
    }
}
```

Test fonctionnel :

Objectif : Identique au précédent. Mais cette fois-ci, il faut s'assurer d'avoir le nouvel affichage.

1. Une fois arrivé dans la liste des restaurants (capture d'écran gauche), cliquez sur l'un de ces derniers. Vous êtes censé arriver sur la vue crée précédemment (capture d'écran de droite).





2. Si vous cliquez sur le bouton « Retour » vous revenez sur l'écran précédent. (capture d'écran de gauche)

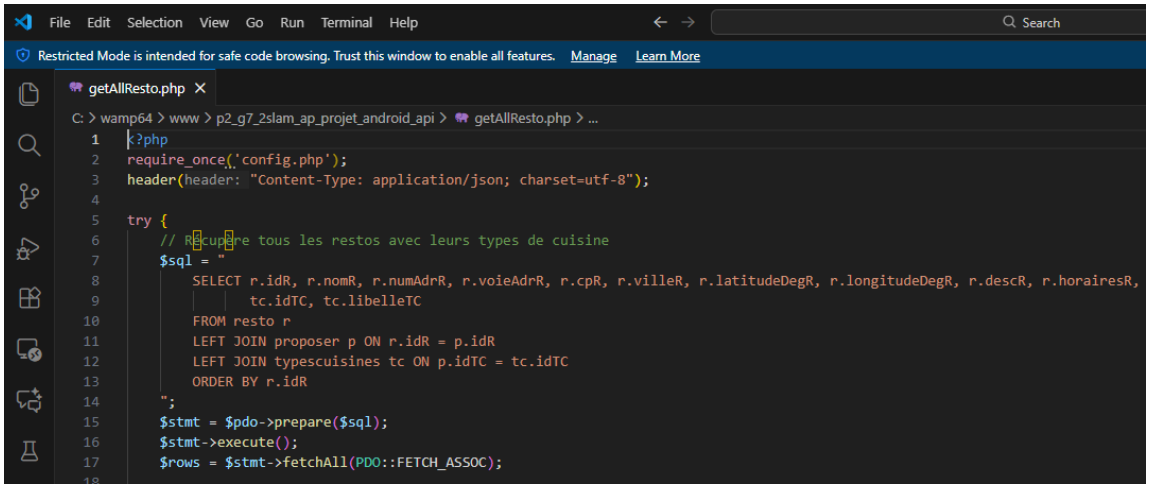
Ticket 3 - API avec récupération des données:

Branche utilisée au moment du ticket (Application Android) : iteration2_ticket3

Programmes (1/2) :

J'ai modifié le fichier getAllResto :

Ce code PHP inclut la configuration de la base de données avec require_once('config.php') et définit l'en-tête pour renvoyer du JSON. Il prépare une requête SQL qui récupère tous les restaurants avec leurs informations et les types de cuisine associés via des LEFT JOIN. La requête est préparée et exécutée avec PDO pour sécuriser l'accès à la base. Enfin, tous les résultats sont récupérés sous forme de tableau associatif grâce à $stmt->fetchAll(PDO::FETCH_ASSOC), prêt à être converti en JSON pour une API.



Puis, cette partie du code transforme les résultats bruts de la base de données en une structure plus exploitable pour une API. Elle commence par créer un tableau vide $restos, puis parcourt toutes les lignes $rows. Pour chaque ligne, elle récupère l'ID du restaurant et vérifie si ce restaurant existe déjà dans $restos. Si ce n'est pas le cas, il l'ajoute avec ses informations principales comme le nom, l'adresse, la latitude, la longitude et la description. Les horaires, qui sont stockés au format HTML, sont convertis en JSON : le code crée un objet DOMDocument, charge le HTML des horaires, parcourt chaque ligne <tr> et extrait les trois colonnes (type, semaine, weekend) pour construire un tableau associatif structuré. Ce tableau JSON des horaires est ensuite ajouté au restaurant. Enfin, le restaurant est inséré dans $restos avec un champ vide « typesCuisines » qui pourra être rempli plus tard avec les types de cuisine associés. Cette logique permet d'avoir une structure claire, unique par restaurant, facile à manipuler et à renvoyer en JSON.

```php

    $restos = [];
    foreach ($rows as $row) {
        $id = $row['idR'];
        // Si le resto n'existe pas encore dans le tableau, on l'ajoute
        if (!isset($restos[$id])) {
            // Transformer les horaires HTML en JSON
            $horairesHTML = $row['horairesR'];
            $horairesJSON = [];
            if (!empty($horairesHTML)) {
                libxml_use_internal_errors(use_errors: true);
                $doc = new DOMDocument();
                $doc->loadHTML(source: $horairesHTML);
                $trs = $doc->getElementsByTagName(qualifiedName: 'tr');
                foreach ($trs as $tr) {
                    $tds = $tr->getElementsByTagName('td');
                    if ($tds->length === 3) {
                        $type = trim(string: $tds->item(0)->textContent);
                        $semaine = trim(string: $tds->item(1)->textContent);
                        $weekend = trim(string: $tds->item(2)->textContent);
                        $horairesJSON[$type] = [
                            "Semaine" => $semaine,
                            "Week-end" => $weekend
                        ];
                    }
                }
            }

            $restos[$id] = [
                "idR" => $id,
                "nomR" => $row['nomR'],
                "numAdrR" => $row['numAdrR'],
                "voieAdrR" => $row['voieAdrR'],
                "cpR" => $row['cpR'],
                "villeR" => $row['villeR'],
                "latitudeDegR" => $row['latitudeDegR'],
                "longitudeDegR" => $row['longitudeDegR'],
                "descR" => $row['descR'],
                "horairesR" => $horairesJSON,
                "typesCuisines" => []
            ];
        }
```

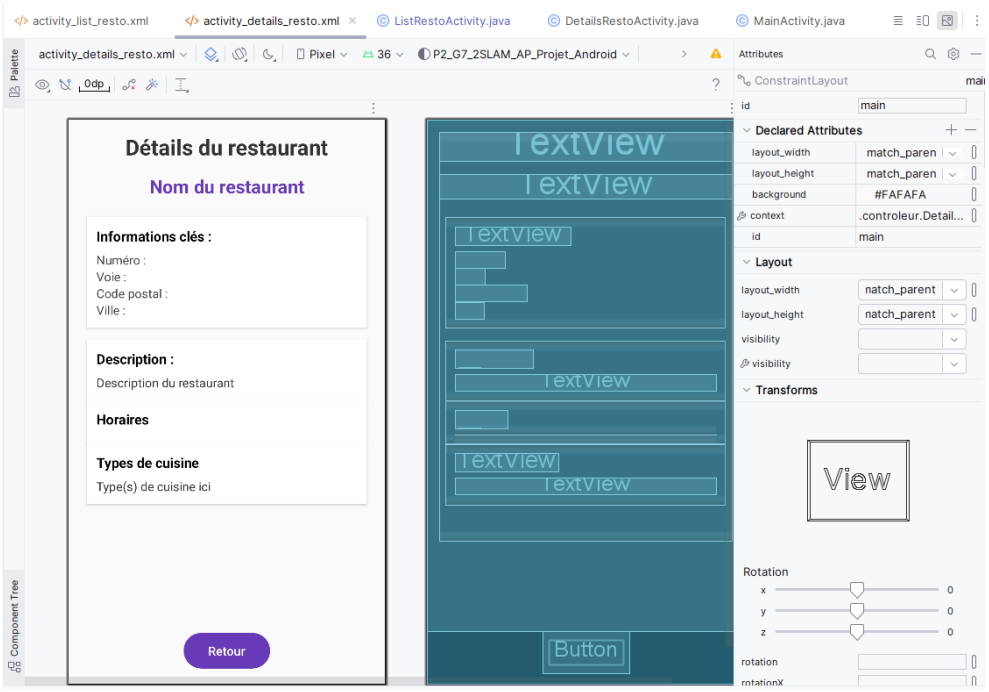
Et pour finir, cette dernière partie du code complète la construction du tableau $restos en ajoutant les types de cuisine pour chaque restaurant. Si la ligne actuelle contient un idTC non vide, un tableau avec l'ID et le libellé du type de cuisine est ajouté au champ typesCuisines du restaurant correspondant. Ensuite, le script prépare la réponse HTTP : il définit le code 200 pour indiquer le succès, puis encode en JSON un tableau contenant le statut "success", le nombre de restaurants et les données elles-mêmes (array_values($restos) réindexe le tableau pour éviter des clés non consécutives). Le JSON est formaté de manière lisible avec JSON_PRETTY_PRINT et conserve les caractères Unicode avec JSON_UNESCAPED_UNICODE. Si une erreur PDO survient, elle est capturée par le bloc catch, le code HTTP est passé à 500, et un message JSON d'erreur est renvoyé avec le détail de l'exception. Cela permet d'avoir une API robuste qui renvoie toujours une réponse structurée, même en cas d'erreur.

```php

        // Ajouter le type de cuisine s'il existe
        if (!empty($row['idTC'])) {
            $restos[$id]['typesCuisines'][] = [
                "idTC" => $row['idTC'],
                "libelleTC" => $row['libelleTC']
            ];
        }
    }

    http_response_code(response_code: 200);
    echo json_encode(value: [
        "status" => "success",
        "count" => count(value: $restos),
        "data" => array_values(array: $restos)
    ], flags: JSON_PRETTY_PRINT | JSON_UNESCAPED_UNICODE);

} catch (PDOException $e) {
    http_response_code(response_code: 500);
    echo json_encode(value: [
        "status" => "error",
        "message" => "Erreur : " . $e->getMessage()
    ], flags: JSON_PRETTY_PRINT | JSON_UNESCAPED_UNICODE);
}
```

Ensuite j'ai ajouté un accesseur et un mutateur pour le idR pour pouvoir afficher les détails d'un restaurant :

```
public int getIdR() {  1 usage
    return idR;
}

public void setIdR(int idR) {  no usages
    this.idR = idR;
}
```

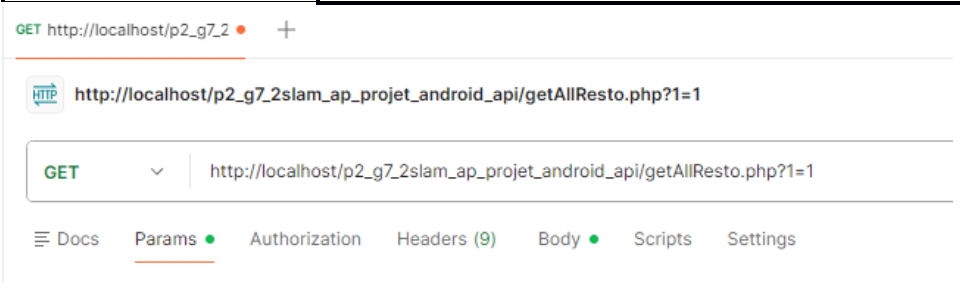
J'ai juste remarqué qu'il manquait un endroit pour afficher les horaires et un autre pour les types de cuisine. Ainsi, j'ai ajouté des TextView qui vont permettre d'afficher les horaires et les types de cuisine.



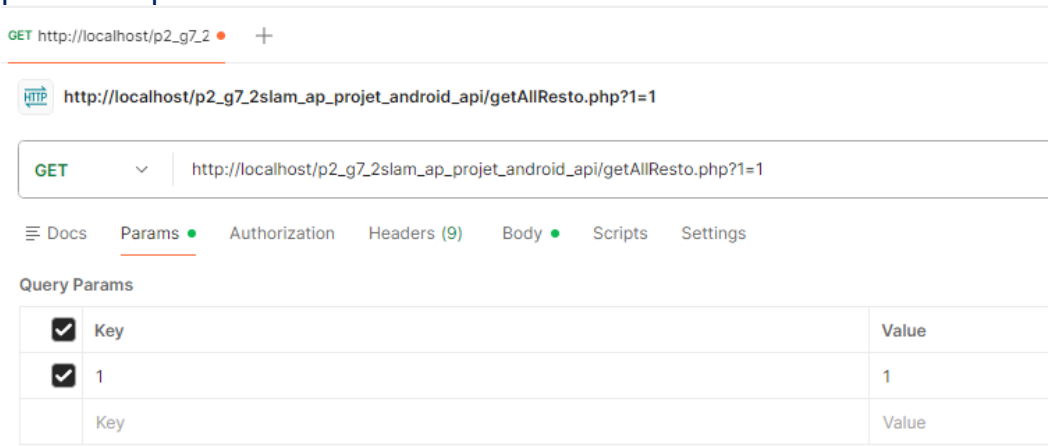
Test fonctionnel :

Objectif : Vérifier que les données sont bien récupérées par l'API.

Si je teste avec Postman, j'ai utilisé la méthode GET pour obtenir les informations d'un seul restaurant. On met ?id=1 dans l'URL car l'API utilise la méthode **GET**, qui transmet les paramètres directement dans l'adresse. Le script PHP récupère cette valeur avec $_GET['id'], ce qui lui permet de savoir quel restaurant renvoyer. Le ? introduit les paramètres, id est la clé et 1 sa valeur. C'est le fonctionnement standard des API REST pour demander une ressource précise.

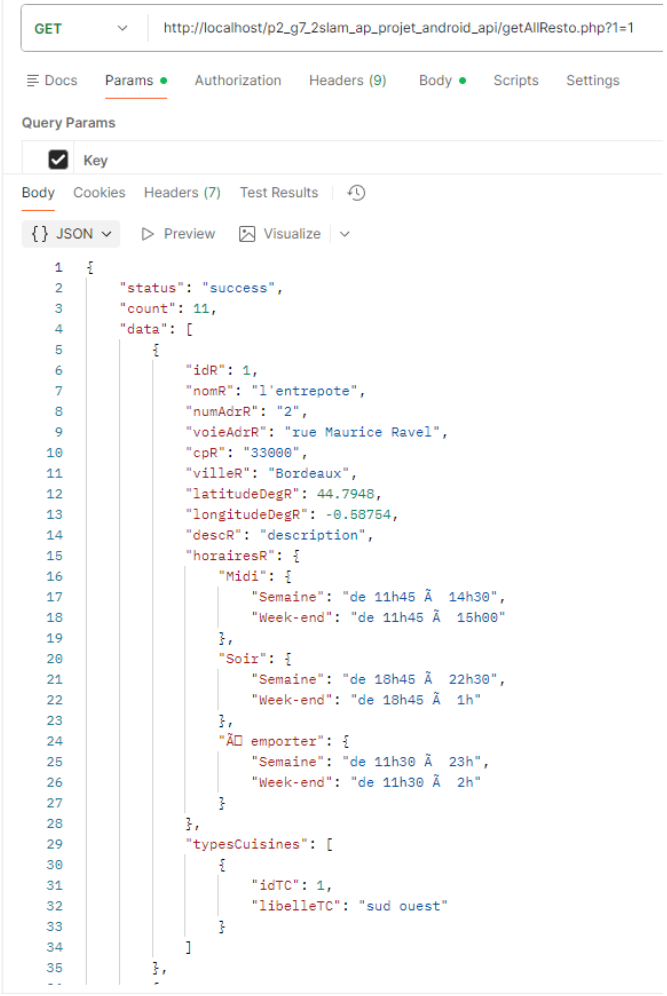


C'est ici dans l'onglet « Params » que j'ai rentré le numéro du restaurant que je souhaite passer en paramètre :



Ensuite je peux faire « Send » et quand je clique, Postman envoie une requête HTTP (ici un **GET**) à l'URL de notre API, avec les paramètres renseignés dans l'onglet **Params**. Le serveur PHP reçoit la requête, lit le paramètre id via $_GET['id'], exécute la requête SQL correspondante et récupère le restaurant demandé. Ensuite, le serveur renvoie une réponse JSON contenant soit les données, soit un message d'erreur. Postman affiche ce JSON dans la fenêtre de réponse.

Et voici la réponse JSON obtenue :



On obtient toutes les informations du restaurant 1

Affichage des lignes 0 - 0 (total de 1, traitement en 0,0006 seconde(s).)

```
SELECT r.idR, r.nomR, r.numAdrR, r.voieAdrR, r.cpR, r.villeR, r.latitudeDegR, r.longitudeDegR, r.descR, r.horairesR,
tc.idTC, tc.libelleTC FROM resto r LEFT JOIN proposer p ON r.idR = p.idR LEFT JOIN typescuisines tc ON p.idTC =
tc.idTC WHERE r.idR = 1;
```

Profilage [Éditer en ligne] [Éditer] [Expliquer SQL] [Créer le code source PHP] [Actualiser]

Tout afficher | Nombre de lignes : 25 | Filtrer les lignes: Chercher dans cette table

Options supplémentaires

idR	nomR	numAdrR	voieAdrR	cpR	villeR	latitudeDegR	longitudeDegR	descR	horairesR	idTC	libelleTC
1	l'entrepote	2	rue Maurice Ravel	33000	Bordeaux	44.7948	-0.58754	description	<table> <thead> <tr> <th>O...	1	sud ouest

On peut voir que c'est la même chose en vérifiant avec une requête SQL.

```sql
SELECT r.idR, r.nomR, r.numAdrR, r.voieAdrR, r.cpR, r.villeR, r.latitudeDegR,
r.longitudeDegR, r.descR, r.horairesR, tc.idTC, tc.libelleTC
FROM resto r
LEFT JOIN proposer p ON r.idR = p.idR
LEFT JOIN typescuisines tc ON p.idTC = tc.idTC
WHERE r.idR = 1;
```

Programmes (2/2) :

Après les tests on peut voir que cela marche donc je peux le rajouter dans le dépôt Gitlab de l'API.

J'ai également créé une méthode pour permettre de transmettre les informations pour Tom qui doit afficher les informations dans la vue :

```java
    private void afficherDetailsRestaurant() {  1 usage
        ArrayAdapter<Restaurant> adapter = new ArrayAdapter<>( context: this,
                android.R.layout.simple_list_item_1, lesRestos);
        listResto.setAdapter(adapter);
        listResto.setOnItemClickListener(( AdapterView<?> parent,  View view,  int position,  long id) -> {
            Restaurant resto = lesRestos.get(position);

            // Afficher toast avec nom et types de cuisine
            Toast.makeText( context: ListRestoActivity.this,
                    text: "Sélectionné : " + resto.getNomR() + "\nTypes : " +
                            (resto.getTypesCuisines().isEmpty() ? "Non renseigné" : String.join( delimiter: ", ",
                                    resto.getTypesCuisines())),
                    Toast.LENGTH_SHORT).show();

            // Ouvrir l'activité détails
            Intent intent = new Intent( packageContext: ListRestoActivity.this, DetailsRestoActivity.class);
            intent.putExtra( name: "restaurant", resto);
            startActivity(intent);
        });
    }
}
```

Le code affiche la liste des restaurants en créant un ArrayAdapter qui utilise la liste lesRestos et l'associe au ListView listResto. Chaque élément de la liste affiche le nom du restaurant (grâce à la méthode toString() de l'objet Restaurant). Un OnItemClickListener est ajouté pour gérer les clics : lorsque l'utilisateur sélectionne un restaurant, l'objet Restaurant correspondant est récupéré, un Toast affiche son nom et ses types de cuisine (ou "Non renseigné" si aucun type n'est défini), puis un Intent est créé pour ouvrir DetailsRestoActivity. L'objet Restaurant est passé à l'activité de détail via putExtra, permettant à celle-ci de charger toutes les informations du restaurant sélectionné.

Ticket 4 : Logique : Affichage des données reçues

Branche utilisée au moment du ticket (Application Android): iteration2_ticket4

Programmes :

J'ai ajouté des dépendances pour les horaires :

```
dependencies {
    // OkHttp (librairie pour requêtes HTTP)
    implementation("com.squareup.okhttp3:okhttp:5.3.0")
    implementation("org.jsoup:jsoup:1.16.1")
```

J'ai complété la classe DetailsRestoActivity, cette activité affiche les détails d'un restaurant sélectionné depuis la liste. Lors du onCreate, elle charge son layout puis récupère toutes les vues (TextView, Button, LinearLayout). Elle récupère ensuite l'objet Restaurant passé depuis l'activité précédente via l'intent. Les informations du restaurant (nom, adresse, description, types de cuisine) sont affichées dans les TextView correspondants. Les types de cuisine sont convertis en texte lisible (séparés par des virgules). Les horaires, stockés sous forme de Map, sont affichés dynamiquement : pour chaque type d'horaire (midi, soir…), un TextView est créé et ajouté au LinearLayout. Un bouton "Retour" permet de fermer l'activité avec finish(). En cas d'erreur (restaurant non reçu), un message Toast s'affiche et l'activité se termine.

DetailsRestoActivity.java

```
16
17  public class DetailsRestoActivity extends AppCompatActivity {
18
19      private Button buttonRetour;  2 usages
20      private TextView textViewNom, textViewNum, textViewVoie, textViewCP, textViewVille,
21              textViewDesc, textViewTypes;  2 usages
22      private LinearLayout layoutHoraires;  4 usages
23
24      private static final String TAG = "DetailsRestoActivity";  1 usage
25
26      @Override
27      protected void onCreate(Bundle savedInstanceState) {
28          super.onCreate(savedInstanceState);
29          setContentView(R.layout.activity_details_resto);
30
31          // Initialisation des vues
32          textViewNom = findViewById(R.id.textViewNomResto);
33          textViewNum = findViewById(R.id.textViewNumAdr);
34          textViewVoie = findViewById(R.id.textViewVoieAdr);
35          textViewCP = findViewById(R.id.textViewCPAdr);
36          textViewVille = findViewById(R.id.textViewVilleAdr);
37          textViewDesc = findViewById(R.id.textViewDescResto);
38          textViewTypes = findViewById(R.id.textViewTypes);
39          layoutHoraires = findViewById(R.id.layoutHoraires);
40          buttonRetour = findViewById(R.id.buttonRetour);
```

Ensuite à la suite, cette partie configure le bouton retour pour fermer l'activité lorsqu'on clique dessus avec finish(). Ensuite, elle récupère l'objet Restaurant envoyé depuis l'activité précédente via l'Intent grâce à getSerializableExtra("restaurant"). Si le restaurant existe, la méthode afficherDetails(resto) est appelée pour remplir les vues avec ses informations. Sinon, un Toast s'affiche pour prévenir l'utilisateur qu'aucun restaurant n'est disponible.

```
        // Bouton retour
        buttonRetour.setOnClickListener( View v -> finish());

        // Récupération du restaurant passé en Intent
        Restaurant resto = (Restaurant) getIntent().getSerializableExtra( name: "restaurant");
        if (resto != null) {
            afficherDetails(resto);
        } else {
            Toast.makeText( context: this,  text: "Aucun restaurant à afficher", Toast.LENGTH_SHORT).show(
        }
    }

```

Puis j'ai fait une méthode afficherDetails, où cette méthode remplit les vues avec les informations du restaurant passé en paramètre. Elle affiche le nom, l'adresse complète (numéro, voie, code postal, ville) et la description dans les TextView correspondants. Les types de cuisine sont affichés séparés par des virgules, ou un texte par défaut si aucun type n'est renseigné. Pour les horaires, le `LinearLayout` est d'abord vidé, puis chaque type d'horaire (avec Semaine et Week-end) est ajouté dynamiquement sous forme de `TextView`. Si aucun horaire n'est disponible, un texte indiquant "Horaires non disponibles" est affiché.

```
    private void afficherDetails(Restaurant resto) {  1 usage
        try {
            textViewNom.setText(resto.getNomR());
            textViewNum.setText("Numéro : " + resto.getNumAdrR());
            textViewVoie.setText("Voie : " + resto.getVoieAdrR());
            textViewCP.setText("Code postal : " + resto.getCpR());
            textViewVille.setText("Ville : " + resto.getVilleR());
            textViewDesc.setText(resto.getDescR());

            if (resto.getTypesCuisines() != null && !resto.getTypesCuisines().isEmpty()) {
                textViewTypes.setText(String.join( delimiter: ", ", resto.getTypesCuisines()));
            } else {
                textViewTypes.setText("Type(s) non renseigné(s)");
            }

            // Affichage des horaires
            layoutHoraires.removeAllViews();
            if (resto.getHorairesR() != null && !resto.getHorairesR().isEmpty()) {
                for (Map.Entry<String, Map<String, String>> entry : resto.getHorairesR().entrySet()) {
                    String type = entry.getKey();
                    Map<String, String> h = entry.getValue();
                    TextView tv = new TextView( context: this);
                    tv.setText(type + " : Semaine " + h.get("Semaine") + ", Week-end " + h.get("Week-end"));
                    tv.setTextSize(16f);
                    layoutHoraires.addView(tv);
                }
            } else {
                TextView tv = new TextView( context: this);
                tv.setText("Horaires non disponibles");
                layoutHoraires.addView(tv);
```

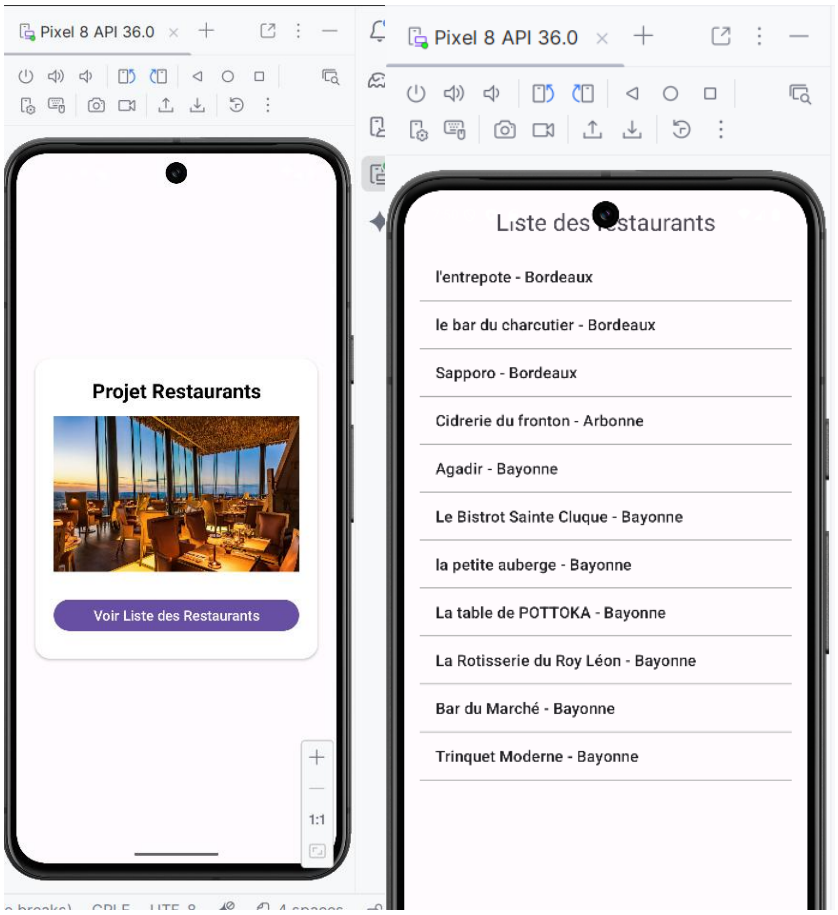
Et je gère ensuite les erreurs lors de l'affichage des détails. Si une exception survient, elle est enregistrée dans le log avec Log.e pour faciliter le débogage, et un Toast informe l'utilisateur qu'une erreur s'est produite lors de l'affichage des informations du restaurant :

```
84              }
85
86          } catch (Exception e) {
87              Log.e(TAG,  msg: "Erreur affichage détails", e);
88              Toast.makeText( context: this,  text: "Erreur affichage détails", Toast.LENGTH_LONG).show();
89          }
90      }
91  }
92
```

Test fonctionnel :

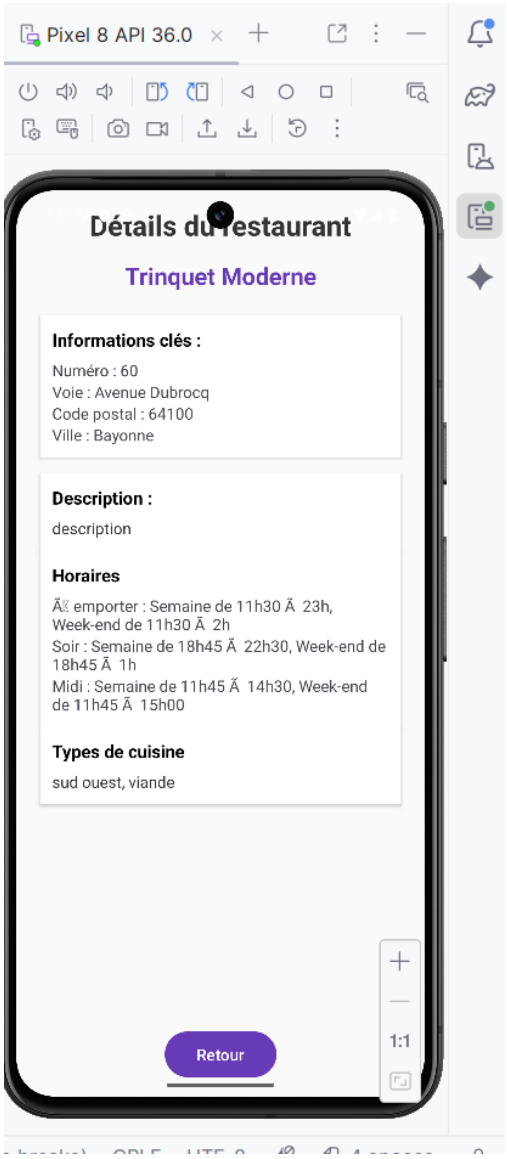
Objectif : Afficher les données récupérées depuis l'API sur l'écran « Détails du restaurant ».

1. On démarre l'application, puis on clique sur Voir Liste des Restaurants :

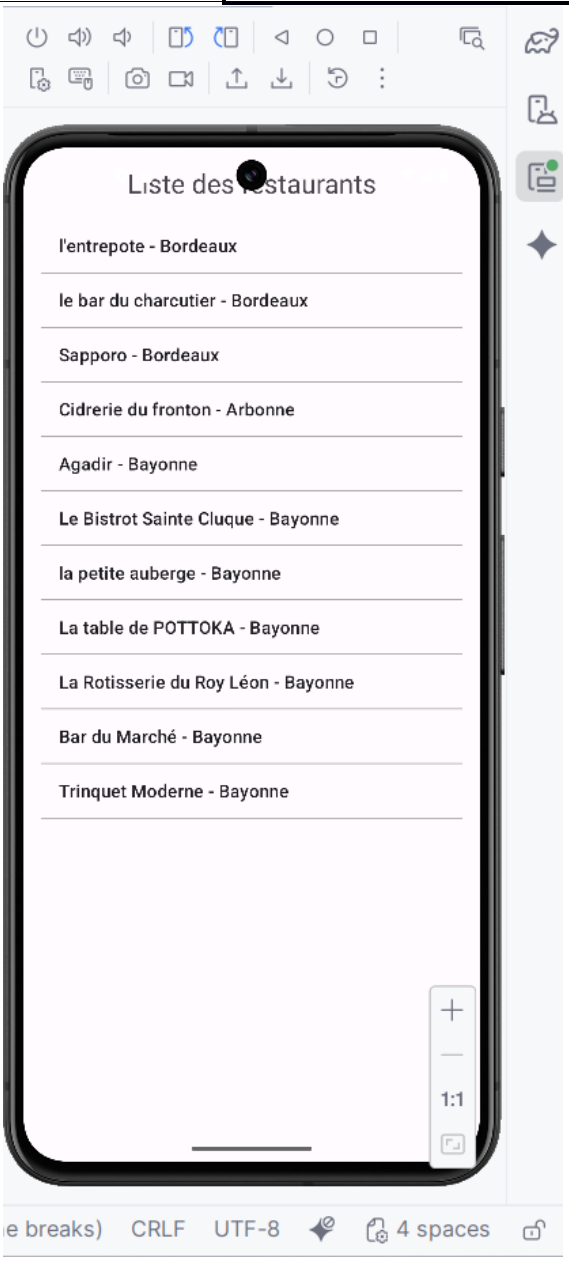


2. On clique sur un des restaurants et cela nous affiche toujours le Toast.

3. Et cela nous emmène vers la page Details avec toutes les infos d'affichés récupérés grâce à l'API :



4. Et le bouton Retour nous ramène bien vers la liste des restaurants :


Liste des restaurants
l'entrepote - Bordeaux
le bar du charcutier - Bordeaux
Sapporo - Bordeaux
Cidrerie du fronton - Arbonne
Agadir - Bayonne
Le Bistrot Sainte Cluque - Bayonne
la petite auberge - Bayonne
La table de POTTOKA - Bayonne
La Rotisserie du Roy Léon - Bayonne
Bar du Marché - Bayonne
Trinquet Moderne - Bayonne
1:1
e breaks)
CRLF
UTF-8
4 spaces

Ticket 5 : Utiliser l'utilisateur resto_util dans l'API

Branche utilisée au moment du ticket (API) : iteration2_ticket5

Programmes :

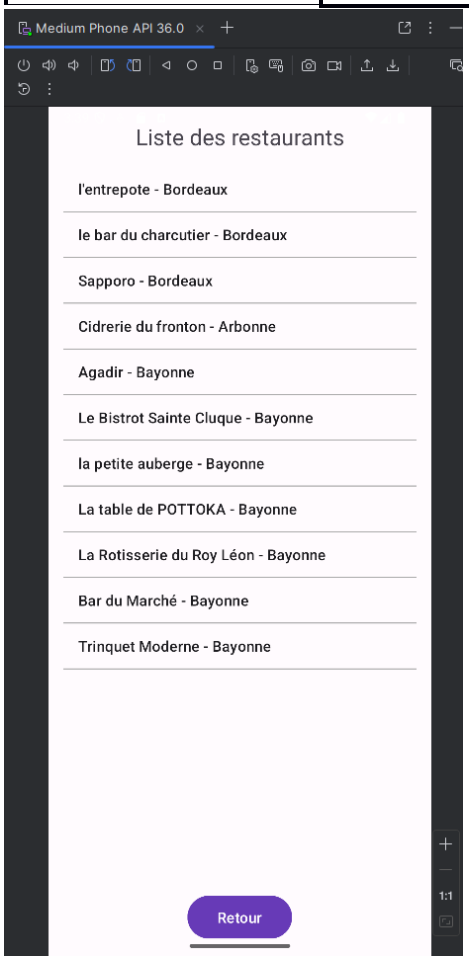
En fait, comme l'utilisateur resto_util a été crée lors du projet d'AP précédent, je le réutilise. Il me reste donc qu'à modifier le nom de l'utilisateur et son mot de passe dans le fichier « config.php ».

```php
<?php
$host = "localhost";
$port = "3306"; // Port MySQL de Wamp
$dbname = "resto2";
$username = "resto_util";
$password = "secret";
```

Test fonctionnel

Objectif : Vérifier que la connexion fonctionne avec le nouvel utilisateur.

1. Lancer l'application et cliquer sur le bouton « Voir Liste des Restaurants ». Normalement nous obtenons l'affichage suivant.


Medium Phone API 36.0
Liste des restaurants
l'entrepote - Bordeaux
le bar du charcutier - Bordeaux
Sapporo - Bordeaux
Cidrerie du fronton - Arbonne
Agadir - Bayonne
Le Bistrot Sainte Cluque - Bayonne
la petite auberge - Bayonne
La table de POTTOKA - Bayonne
La Rotisserie du Roy Léon - Bayonne
Bar du Marché - Bayonne
Trinquet Moderne - Bayonne
Retour
1:1

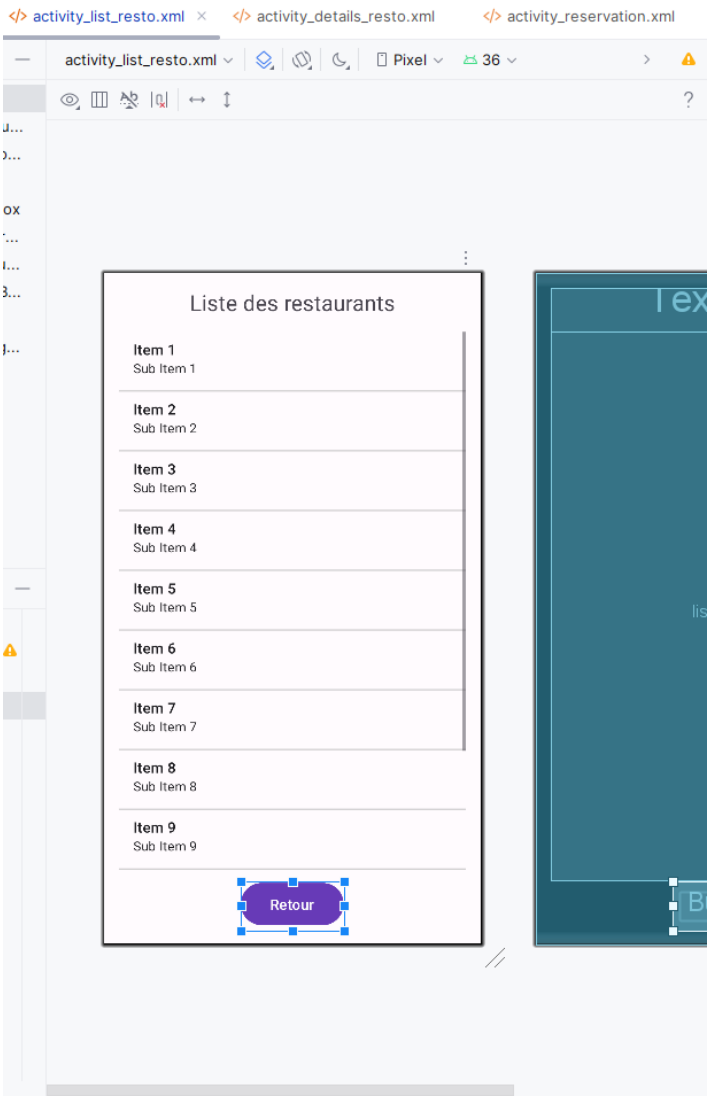
2025/2026	AP – Projet 2 – Android – Itération 2
BTS SIO 2SLAM	Auteur : PORTOLLEAU Anaïs & DELAUNAY-GUITTON Benjamin & CHARBONNEAU Tom
	Date de rédaction : 13/11/2025

Ticket 6 : UI – Améliorer la navigation entre les vues

Branche utilisée au moment du ticket (Application Android) : itération2_ticket6

Programmes :

J'ai bien amélioré la navigation entre les vues pour permettre de passer de la vue de la liste des restaurants à la vue d'accueil :



Et également le code :

J'ai ajouté

```
private Button buttonRetour;   // <<<<<<<<< AJOUT  2 usages
```

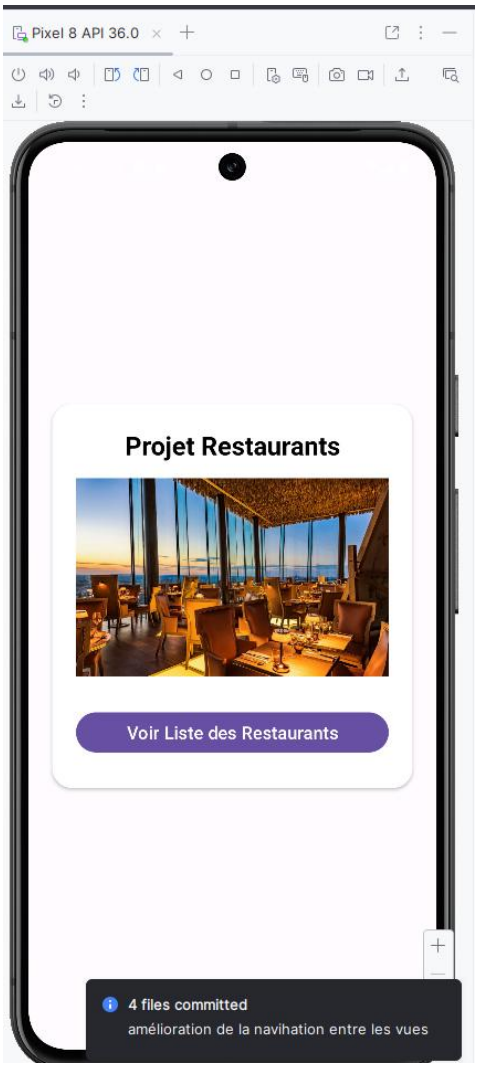
j'ai rajouté également du code dans :

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_list_resto);

    listResto = findViewById(R.id.listResto);
    buttonRetour = findViewById(R.id.buttonRetour); // <<<<<<<<< AJOUT

    // Listener du bouton Retour
    buttonRetour.setOnClickListener( View v -> finish()); // <<<<<<<<< AJOUT

    chargerLesRestaurants();
}
```

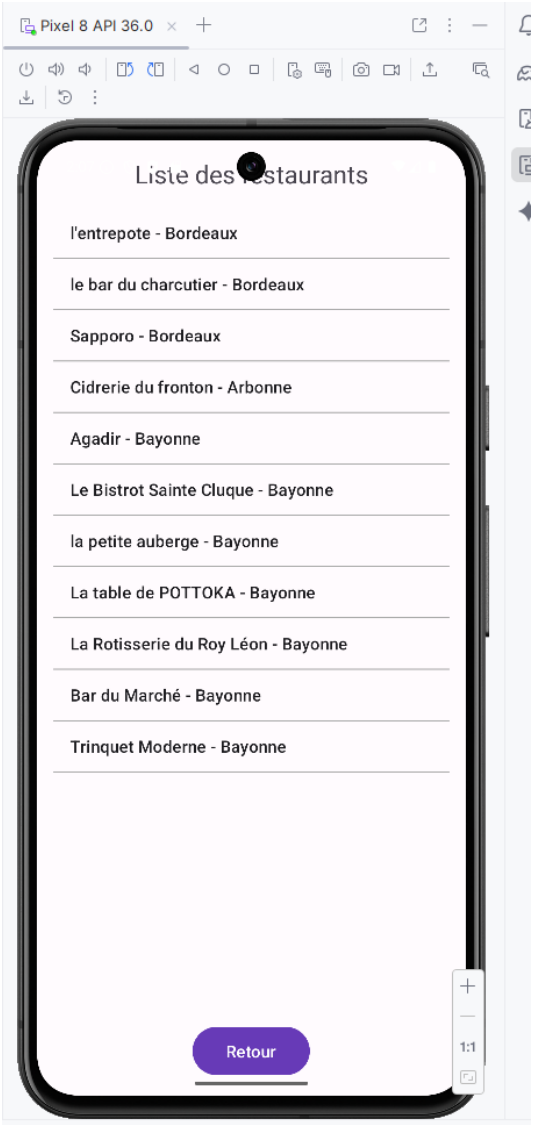
Test fonctionnel :

Objectif : Vérifier le bon fonctionnement de la navigation entre l'écran « Liste des restaurants » et « Accueil ».

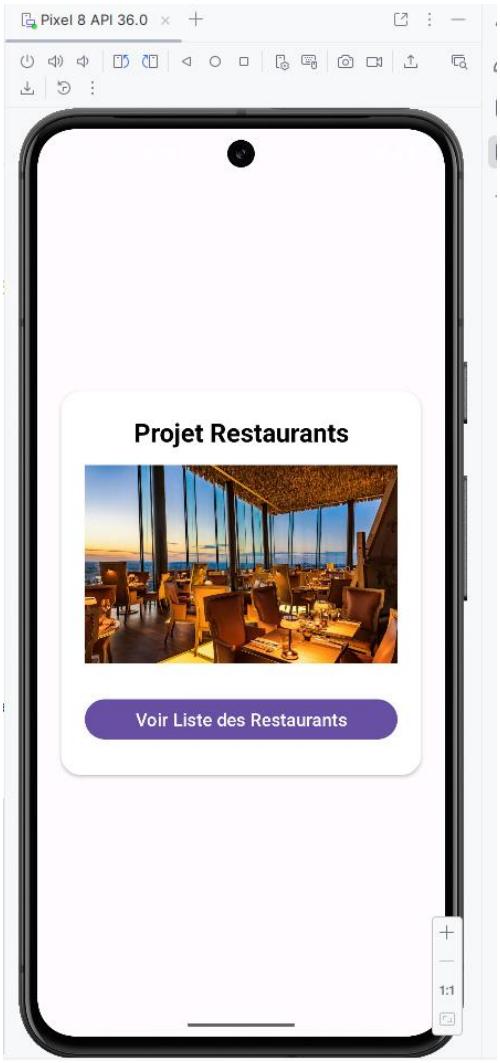
1. On arrive toujours sur la page d'accueil :



2. Puis on fait voir Liste des restaurants et on peut voir que le bouton apparait bien :



3. Et si on fait Retour cela revient bien vers la page d'accueil :



Ticket 7 : Documenter les projets

Branche utilisée au moment du ticket (Application Android) : iteration2_ticket7
Branche utilisée au moment du ticket (Projet API) : iteration2

Programmes : Projet Android

Je me suis occupé de commenter le code existant afin d'expliquer l'utilité des classes ou des méthodes comme ici dans la classe ListRestoActivity en voici un petit aperçu :

```java
// Activité principale qui récupère la liste des restaurants depuis une API et les affiche
public class ListRestoActivity extends AppCompatActivity {

    // Composant graphique (Vue) pour afficher la liste défilante
    3 usages
    private ListView listResto;

    // Liste qui contiendra les objets "Restaurant" créés à partir du JSON
    4 usages
    private ArrayList<Restaurant> lesRestos = new ArrayList<>();

    // Client HTTP (OkHttp) pour effectuer les requêtes réseau
    1 usage
    private OkHttpClient httpClient = new OkHttpClient();

    3 usages
    private static final String TAG = "ListRestoActivity";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_list_resto);

        // Liaison avec la ListView du layout
        listResto = findViewById(R.id.listResto);

        // Lancement de la récupération des données dès le chargement de l'écran
        chargerLesRestaurants();
    }

    // Méthode gérant la requête HTTP vers l'API
    1 usage
```

Programmes : Projet API

De même pour l'api avec le fichier « getAllResto.php ».

```php
try {
    $restos = []; // Stocke les restaurants dans un tableau

    // Parcourt les lignes de la requête
    foreach ($rows as $row) {
        $id = $row['idR'];

        // Si le restaurant n'est pas déjà dans le tableau, on l'ajoute
        if (!isset($restos[$id])) {

            // La BDD stocke du HTML (<table>). On le parse ici pour envoyer des données structurées à l'Android.
            $horairesHTML = $row['horairesR'];
            $horairesJSON = []; // Tableau des horaires sans les balsies html
            if (!empty($horairesHTML)) {
                libxml_use_internal_errors(true); // Masque les erreurs de validation HTML
                $doc = new DOMDocument();
                $doc->loadHTML($horairesHTML);
                $trs = $doc->getElementsByTagName('tr');

                // Extraction des données cellule par cellule (td)
                foreach ($trs as $tr) {
                    $tds = $tr->getElementsByTagName('td');

                    // On vérifie que la ligne contient bien 3 cellules
                    if ($tds->length === 3) {
                        $type = trim($tds->item(0)->textContent);
                        $semaine = trim($tds->item(1)->textContent);
                        $weekend = trim($tds->item(2)->textContent);

                        // On ajoute les horaires au tableau
                        $horairesJSON[$type] = [
                            "Semaine" => $semaine,
                            "Week-end" => $weekend
                        ];
                    }
                }
            }
```

Ticket 8 : Ajout des scripts SQL dans le projet Android

Branche utilisée au moment du ticket (Application Android) : iteration2_ticket8

Programmes :

J'ai simplement ajouté un nouveau package intituler « sql » dans lequel on retrouve le script pour créer l'utilisateur avec les droits qu'on lui accorde sur la BDD et le script de la BDD. Il n'y a pas de test fonctionnel pour cette partie car elle a été réalisée dans l'itération 1. De plus, il est possible de suivre les instructions dans le README de l'itération 2.