## Exercise 5.1: Fetch recipes via a Rest API

There is a back-end server to accompany the Recipe Book application. You can access via <http://localhost:3000>. Use the documentation at the default endpoint to get familiar with the API. To execute requests, you need to be authenticated. Press the authorize button and input any username and an empty password. Extend your application to fetch the recipe list from the server.

Steps:

* Add the dependency for the OkHttp library as shown in the lecture.
* Add a String property to the Recipe class called *id*. Create appropriate constructors for the class to work easily created objects with only some of the parameters.
* Remove the hardcoded recipes from the RecipeCollection constructor. Add the following method to the class:

public void setRecipes(List<Recipe> recipes) {

this.recipes.clear();

this.recipes.addAll(recipes);

}

It is done like this, so that we don’t need to call *setData()* on the adapter every time we fetch the recipes from the database.

* Write a method to make an HTTP request using OkHttp to fetch a list of all available recipes. Execute this method in *onCreate()* of the RecipeListActivity. Add a refresh menu button that calls the method when pressed.
* The result of the requests is in JSON format. Use *JSONArray* and *JSONObject* to parse the data and populate the RecipeCollection via the *setRecipes()* method.

Remarks:

* Take care of the required permissions for your app.
* Use the trick from the lecture to be able to access the network from the UI thread. When running on a real device: Do you notice a lag?
* Since authorization is required, we are going use the device’s unique ANDROID\_ID. Create a credentials String as follows and add it as a value to the header parameter with the name *Authorization*:

String credentials = Credentials.basic(Settings.Secure.getString(activity.getContentResolver(), Settings.Secure.ANDROID\_ID), "")

* If the data in the list view does not refresh after updating the data you can force the UI to update the view with either calling *invalidate()* on the list view or *notifyDataSetChanged()* on the adapter.

## Exercise 5.2: Fetch recipe details via the Rest API

* Change the OnItemClickListener of the list view in the RecipeListActivity to send the id value of the Recipe object instead of its index.
* Add another variable to the Recipe class, called *image,* of type byte array. It will be used to store the image of each recipe.
* After starting the RecipeDetailsActivity, make a request to the server to fetch details about the recipe using its id, like in Exercise 5.1. The response object could contain an image encoded as a Base64 string. You need to decode it to a byte array:

String imageBase64 = jsonObject.getString("image");

byte[] decodedString = Base64.decode(imageBase64, Base64.DEFAULT);

and then again to a Bitmap object and set it to the image view*:*

Bitmap bitmap = BitmapFactory.decodeByteArray(recipe.getImage(), 0, recipe.getImage().length);

If the recipe doesn’t have an image, simply display the placeholder.

## Exercise 5.3: Add new recipes to the database via the Rest API

* Write a method that adds a recipe to the database using the POST endpoint.
* Use *MultipartBody.Builder* as your request body. Multipart is used as it allows for images to be send as part of the request. This will be needed for a later exercise.
* If the new Recipe isn’t added to the list view upon going back to the RecipeListActivity, overwrite the *onResume()* method to force the UI to update.

## Exercise 5.4: Search recipes by name (optional)

* In the RecipeListActivity add an EditText field and a button to search recipes by name, like in the movie database example from the lecture.
* In the HttpBuilder of the request add the value from the EditText field as parameter (if it’s not empty) and afterwards perform a new request to the server. The list should now only contain recipes that start with the string from the text field.