Service API

The service API package defines and handles the interaction between the app and external RESTful APIs over HTTP(S). It also performs its own error handling and output parsing.

For use as a RESTful API (instead of native Java code, detailed below), more information can be found in the `api` package here.

Interface

The `makeRequest` method takes the service name and required data as parameters and performs the API request, the output is a sentence for the speech synthesiser to speak aloud to the user (with the relevant data included).

- This output is put onto an API response queue, maintained by the main controller of the app.

The recognised service names are:

- *"air quality"*
- *"book"*
- *"charity search"*
- *"charity by city"*
- *"current weather"*
- *"dictionary"*
- *"ingredient"*
- *"joke"*
- *"nearest transport"*
- *"news"*
- *"random recipe"*
- *"recipe"*

- *"recipe instructions"*
- *"stocks"*
- *"transport search"*
- *"weather forecast"*

Getting Responses

The API response queue should have the type:

```
BlockingQueue<ApiResponse> appQueue;
```

Periodically polling this queue returns an `ApiResponse` object returning the service's response after a call is made.

The string response, used in speech synthesis, can be retrieved using the `getResponse()` method of the object.

The name of the service that was called can be retrieved using the `getName()` method of the object.

A service may return (required) data that is not suitable for speech synthesis e.g. an image to display.
This type of data is referred to as *'metadata'*, stored in a `HashMap<String, Object` named `metadata`.
It is retrieved by calling the `metadata()` method of an `ApiResponse` object.

- Note that there is no standard for the representation of data inside the hashmap - this is entirely dependent on the service, and its corresponding `parseOutput` method.
- Expected metadata (for successful API calls) for relevant services is detailed below, **for that specific service only**.

The `getName()` method should be used to check what service was returned to determine how to read its metadata, if required.

An example implementation showing how to retrieve responses can be found in test package.

The following section defines the required parameters needed by each service. See the *"Adding Services"* section for more information on how to integrate new services.

API Formats

The following set of tables define the required data needed by each service. The internal structure of a `RESTful`
service API request is a `HashMap<String, String>` object where the **Attribute** columns below represent the exact field names in
the hash map for a service.

Current Weather API

A service allowing the user to retrieve information about the current weather in their (or explicitly specified) city.

Attribute	Type	Default	Description
`CITY_NAME`	String	`"London"`	The name of the city the user's device is located in.
`COUNTRY_CODE`	String	`"uk"`	The two-character ISO country code of the city. The full list of possible
`LANGUAGE`	String	`"en"`	The two-character (or three) ISO code of the language to return the we

Weather Forecast API

A service returning the predicted weather for the user's exact location.

Attribute	Type	Default	Description
`LAT`	Double	`51.534121`	The latitude of the user's current (or most recent) location.
`LON`	Double	`-0.006`	The longitude of the user's current (or most recent location.
`LANGUAGE`	String	`"en"`	The two-character (or three) ISO code of the language to return the weath
`DAYS`	Integer	`1`	The number of days to forecast for. The minimum is 7 and the maximum

Air Quality API

A service returning the air quality index for a given city.

Attribute	Type	Default	Description
`CITY_NAME`	String	`"London"`	The name of the city the user's device is located in.

Stocks API

A service allowing the user to retrieve stock information on an equity of their choice.

Attribute	Type	Default	Description
FUNCTION	String	"TIME_SERIES_INTRADAY"	The time series of your choice.
SYMBOL	String	"IBM"	The name of the equity.
INTERVAL	Integer	1	Time interval between two consecutive data points in the t

Joke API

A service retrieving a random or specific joke.

Attribute	Type	Default	Description
TERM	String	""	The search term to search for a joke.

Dictionary API

A service retrieving the definition, examples and synonyms for a given word.

Attribute	Type	Default	Description
WORD	String	"hello"	The word to retrieve the definition of.
LANGUAGE	String	"en"	The two-character (or three) ISO code of the language to return t
INCLUDE_SYNONYMS	Boolean	"false"	Indicates whether to retrieve the synonyms of the word as well.
SYNONYMS_ONLY	Boolean	"false"	Allows for thesaurus usage, retrieving only the synonyms of the w

Nearest Transport API

This service returns either the nearest train stations or bus stops for a given location range.

Attribute	Type	Default	Description
MIN-LAT	Double	51.530121	The minimum latitude, representing one corner of the bounding b
MAX-LAT	Double	51.538121	The maximum latitude, representing one corner of the bounding b

Attribute	Type	Default	Description
`MIN-LON`	Double	`-0.009`	The minimum longitude, representing one corner of the bounding
`MAX-LON`	Double	`-0.001`	The maximum longitude, representing one corner of the bounding
`TRANSPORT`	String	`"train_station"`	Indicates whether to search for a train station (`train_station`) o

Note that this service does not return any spoken ouput, just the coordinates of the nearest transport points defined below.

Metadata The metadata returned by this service contains just one top-level field - `locations` which is an array of latitude, longitude pairs for each returned transport point. Each element in the `locations` array is a `Double[]` array, with the following elements:

Attribute	Type	Description
`latitude`	`Double`	The latitude of the transport point.
`longitude`	`Double`	The longitude of the transport point.

Note that the ordering here is important - **the first element in the array is the `latitude`, the second the `longitude`**.

Transport By Search API

This service returns information on a bus stop or train station by its name.

Attribute	Type	Default	Description
`QUERY`	String	`"euston"`	The name of the train station or bus stop to search for.
`TRANSPORT`	String	`"train_station"`	Indicates whether to search for a train station (`train_station`) or

Metadata The metadata returned by this service is:

Attribute	Type	Description
`latitude`	`Double`	The latitude of the transport point.
`longitude`	`Double`	The longitude of the transport point.

Random Recipe API

This service returns a random recipe.

This service requires no parameters - supplying them has no effect.

Metadata The metadata returned by this service is:

Attribute	Type	Description
`recipe-id`	`String`	The ID of the recipe, useful for further API calls, e.g., fetching the full instructions us
`image`	`String`	A URL to the cover image of the recipe.

Recipe By Search API

This service returns a set of recipes by searching via natural language.

Attribute	Type	Default	Description
`QUERY`	String	`""`	The search term.

Metadata The metadata returned by this service is:

Attribute	Type	Description
`recipe-id`	`String`	The ID of the recipe, useful for further API calls, e.g., fetching the full instructions us
`image`	`String`	A URL to the cover image of the recipe.

Recipe By Ingredient API

This service returns a set of recipes with certain ingredients, specified by the user.

Attribute	Type	Default	Description
`INGREDIENTS`	String	`""`	A comma separated string of ingredients to search for.

Metadata The metadata returned by this service is:

Attribute	Type	Description
`recipe-id`	`String`	The ID of the recipe, useful for further API calls, e.g., fetching the full instructions us
`image`	`String`	A URL to the cover image of the recipe.

Recipe Instructions API

This service returns the instructions for the recipes returned by the services above.

Attribute	Type	Default	Description
`ID`	String	`""`	The ID of the recipe.
`DETAILED`	Boolean	`true`	Whether to split instructions into steps (`true`) or return a plain description

Note that this service is intended to be used *internally* - each recipe service above returns the `ID` of the recipe in the `metadata` of the response object; the recipe instructions service should then be used to retrieve the instructions of that recipe.

This service also returns no spoken output. The `metadata`, defined below, can be used to programmatically decide what steps to read (instead of reading it all at once).

Metadata The metadata returned by this service is:

Attribute	Type	Description
`steps`	`ArrayList<String>`	An array of instructions for the recipe - each element is one instruction ste

News API

This service returns a news article based on the user's search.

Attribute	Type	Default	Description
`QUERY`	String	`""`	The search term, in natural language.
`LANGUAGE`	String	`"en"`	The language to return the result in. Uses the two-character IS0-639-1 code sc

Metadata The metadata returned by this service is:

Attribute	Type	Description
`url`	`String`	A URL to the full news article.
`image`	`String`	A URL to the cover image of the article.

Charity Search API

This service returns information on a specified number of charities, based on the user's search in natural language.

Attribute	Type	Default	Description
`QUERY`	String	`""`	The search term, in natural language.
`VALUES`	String	`1`	The (maximum) number of charities to return information on.

Metadata The metadata returned by this service contains just one top-level field: `charities` which is an array of maps (each one representing a charity) where each map contains the following information:

Attribute	Type	Description
`name`	`String`	The name of the charity.
`URL`	`String`	A URL to the charity's registered page.
`donationURL`	`String`	A URL to donate to the charity.
`latitude`	`Double`	The latitude of the charity.
`longitude`	`Double`	The longitude of the charity.

Charity By City API

This service returns information on a specified number of charities in a given city.

Attribute	Type	Default	Description
`CITY`	String	`"london"`	The search term, in natural language.
`VALUES`	Integer	`1`	The (maximum) number of charities to return information on.

Metadata The metadata returned by this service contains just one top-level field: `charities` which is an array of maps (each one representing a charity) where each map contains the following information:

Attribute	Type	Description
`name`	`String`	The name of the charity.
`URL`	`String`	A URL to the charity's registered page.
`donationURL`	`String`	A URL to donate to the charity.
`latitude`	`Double`	The latitude of the charity.
`longitude`	`Double`	The longitude of the charity.

Book By Search API

This service returns information on a book based on the user's search.

Attribute	Type	Default	Description
`QUERY`	String	`""`	The search term, in natural language. Entirely optional.
`TOPIC`	String	`""`	A certain topic to look for. Entirely optional.
`LANGUAGES`	String	`"en"`	A comma separated string containing the list of languages to search for. Uses

Note that both the `QUERY` and `TOPIC` attributes are optional, but at least one must be supplied.

Metadata The metadata returned by this service is:

Attribute	Type	Description
`id`	`Integer`	The ID of the book, useful for making further API calls.
`image/jpeg`	`String`	A URL to the image of the book cover.
`text/html`	`String`	A URL to an online version of the book, formatted with HT
`text/plaintext`	`String`	A URL to an online verison of the book, in plain text with
`application/x-mobipocket-ebook`	`String`	A URL to an ebook download of the book.
`application/epub+zip`	`String`	A URL to an ebook download of the book in a different for

Adding Services

The API service interaction is highly extensible. There are two methods to extending Concierge with new services, detailed below.

JSON Schema

The JSON schema greatly simplifies the extension of Concierge's already rich API interaction ecosystem. It also allows
for new services to be added in a language-independent manner, providing a simple and intuitive, yet powerful interface.

Full details on getting started with the schema can be found here.

- After following the instructions, no code needs to be modified to add the service to Concierge.

For most services, the JSON schema should be sufficient to perform and parse API calls. However, there are some limitations with the schema; to circumvent these, the service can be implemented as a Java class.

Java Class

Should you require more fine-grained control over the API URL, or the response parsing, it is recommended to do so using a Java class.

A service must extend the abstract `ServiceRequest` class, providing the following attributes:

- `URL` - The URL where the resource provided by the API is located.
- If the URL requires any named parameters, they are to be added in the following format:
- *e.g. the Weather API requires a* `lang` *attribute:* `...lang={LANGUAGE}` *as per the* ***Attribute*** `LANGUAGE` *in its API format.*
- `name` - The name of the service - **must be unique**.
- `category` - The category the service falls under.
- `APIKey` - A unique string used to access the service's API - this is obtained by registering for one on the API's website. Pass an empty string if it is not required by the service.

- `payload` - A `HashMap` containing the parameters required to perform the API request. The necessary keys (and values) for each service are defined in the section above.

Note that the concrete service class must only take one parameter - the `payload`. The concrete constructor must be of the form:

```
public NewServiceRequest(HashMap<String, String> payload) {
    super("URL_HERE", "NAME_HERE", "CATEGORY_HERE", "API_Key_HERE", payload);
}
```

It will then be called by the `ServiceFactory` as `new NewServiceRequest(payload);`

It must also implement the following methods:

- `parseOutput(HashMap<String, Object> response);` - Defines how each service interprets its output from the API. This is also where any metadata should be set.
- `String handleErrors(HashMap<String, Object> response);` - Defines how each service interprets error messages from the API.
- `String getErrorCode(HashMap<String, Object> response);` - Defines how the HTTP error code is represented and retrieved for each service. Each service API will have a different way of representing HTTP error codes.
- `HashMap<String, String> populatePayload();` - Inserts default data into the payload if not given to avoid malformed requests. This method is what applies the attributes from the API format to a service.

The service can then be called by adding its name to the switch statement in the `ServiceFactory` by adding a new case for its `name` attribute in lowercase and returning a new object of the service (which takes the `payload` as its only parameter). No other code interaction needs to take place.

The service should be placed in the `services` package, in the relevant category package, defining a new package if it does not fall into an existing category.

Documentation

With either extension method, please ensure to update the documentation accordingly after adding new services.

The new service's name should be add to the bullet list of recognised service names, preferably to maintain alphabetical order.

The new service's API format should also be listed in the "API Format" section with its description, parameters it takes (if applicable) and any metadata it returns.

The new service's name should also be added to the bullet list of endpoints defined in the RESTful API package's README.