Thank you for your interest in B12, and congratulations on making it this far in the interview process! In this coding exercise, you will solve a problem by stitching together data from publicly-available APIs. You should:

- Not spend more than 4 hours on the assignment. It's okay if you don't have a complete solution by the end of that time,
- Use your own development environment and computer,
- Likely use search engines, sources like Stack Overflow, or AI coding agents, which we think is great, and
- Maintain ownership over your code, sharing the .zip file or git repository with us only so that we can review it

Whether or not you choose to leverage AI agents in building your solution, we expect that:
- You fully understand all of the code in your solution and are able to answer questions about its details.
- You thoroughly test your solution and write appropriate unit or integration tests to verify that functionality works as expected (except where the exercise instructs you to avoid spending time on writing tests, see below)

The Problem: Flight Optimization

If you have ever booked a plane flight, you know that the range of available airlines, itineraries, and prices can be overwhelming. Tools like kiwi.com make it easy to search and book flights, and also make their data available for public API usage. In this problem, we'll examine how to find the best flight for an unusual customer who cares about more than just price.

Data sources

To solve this problem, you'll need to have data about flight schedules and locations around the world. Take advantage of the free and publicly available APIs from Kiwi.com, hosted at https://tequila-api.kiwi.com/.
- We've already created a (free) account, so you don't have to register yourself.
- Authenticate all requests using our API key, **`fUfVhV-v87mtISxkJPlasopiB3mmosJ1`**, passing the key with each request using the `apikey` header.
- See the Appendix for relevant documentation for how to use the Kiwi APIs, in particular the locations API for looking up airport codes and cities, and the aggregation search API for finding flights between locations.

Part 1: Find the cheapest flight per kilometer (Python script)

Many flight search tools will show you the cheapest flight between two cities. But our traveler wants to make sure she's getting the best value for the distance. Write a Python script that picks a destination from a list of possible cities, and finds the most affordable flight in ***dollars per***

kilometer between the start and destination airports. This is different from raw cost: for example, a $1000 flight that covers 3000km (0.33 $/km) is preferable to a $200 flight that covers 400km (0.5 $/km).

Your script should be callable as:
`./flight-optimizer --from <city> --to <city> [<city> ...]`, where:

- `--from` specifies the departure city
- `--to` specifies a list of potential destination cities
- `<city>` is the name of a city in string format, e.g. `London` or `Paris`

Your script should output:

- The destination with the best flight (e.g., `Paris`)
- The price of the flight, **in dollars per kilometer** (e.g., `$2.50/km`)

You may assume:

- Your flight can leave any time in the next 24 hours.
- You are buying plane tickets for a single adult.
- You are flying to/from the main airport in each city.
- You can use any consistent metric to compute the distance between two airports. Your programming language of choice might have libraries for computing such distances, e.g. this library for computing Haversine distance in python, and you are also welcome to copy code for a distance function from the internet.

Part 2: Expose your script with an interface

A blog post describing your script got retweeted by a leading venture capital firm, and now everyone is clamoring to use it! Unfortunately, most people on twitter have no engineering background and are incapable of running a script from the command-line. A user-friendly interface accessible from the web would go a long way towards sharing your vision for cost-effective flights with the world. In the language and framework of your choice (at B12, we would use React on the frontend, and Django on the backend), implement a web app that accepts departure and destination cities from the user, calls the flight search logic we've implemented, and displays the best flights.

- Focus on the frontend, not the backend. Don't implement more than a single API call on the server-side to run your script, don't implement any data storage on the backend, and don't write tests for your backend web app code.
- Your frontend should include, at a minimum:
    - An input that allows the user to specify their departure city
    - An input that allows the user to specify a list of destination cities
    - A view to display the best destination and the price of the best flight in dollars per kilometer.
- Include code to easily run your web app so we can try it out (e.g., using a common build framework like webpack).

- In a `.txt` file, document your web app. What software needs to be installed to run it? How can we run the app and test it?

Appendix: Kiwi API documentation

The following is a subset of the Kiwi API documentation. Not all calls are necessary to complete the exercise. **All endpoints can be accessed at https://tequila-api.kiwi.com/, using the API key in the 'Data sources' section above.**

Locations API

This document describes the Kiwi.com locations search API.
Locations is a simple API used to search, suggest and resolve locations in various situations.
The responses are G-zipped and need to be unpacked (response header Content-Encoding: gzip).

GET /locations/query

Search by query. Type of request used mainly for suggestions (based on incomplete names)

Parameters

Name	Description
apikey * `string` *(header)*	use the API key in the 'Data sources' section above.
term * `string` *(query)*	searched term (for suggestions). This parameter expects a full IATA code. If IATA code is not given, the search will go through other available fields: 'id', 'name' or 'code' of the location. It also depends on the 'location_types' specified eg. airport, city, country. The search that is used behind the scenes is elasticsearch. It returns data based on relevancy and many other factors. *Example* : PRG
locale `string` *(query)*	desired locale output - this is the language of the results. Should any other locale be used other than the specified locales, en-US is used. *Available values* : ar-AE, cs-CZ, da-DK, de-DE, el-GR, en-US, es-ES, fi-FI, fr-FR, hu-HU, is-IS, it-IT, iw-IL, ja-JP, ko-KR, lt-LT, ms-MY, nl-NL, no-NO, pl-PL, pt-BR, pt-PT, ro-RO, ru-RU, sk-SK, sr-RS, sv-SE, th-TH, tr-TR, uk-UA, zh-CN, zh-TW *Example* : en-US

location_types `string` *(query)*	desired location output, accepted values: `station`, `airport`, `bus_station`, `city`, `autonomous_territory`, `subdivision`, `country`, `region`, `continent`. To use more than one `location_types`, use multiple `&location_types=` *Example* : airport
limit `integer` *(query)*	default value = 10. Desired number of results in the output. *Example* : 10
active_only `boolean` *(query)*	default value = true. It displayes all active locations. *Example* : true
sort `string` *(query)*	desired order of the output. For A->Z use 'sort=name', for Z->A use 'sort=-name'. Sorting by other fields, e.g. ‘sort=rank’, is possible.

Responses

Code	Description

200	**OK**

Media type: application/json

Example Value


```
{
  "locations": [
    {
      "id": "LCY",
      "int_id": "9625",
      "active": true,
      "code": "LCY",
      "name": "London City Airport",
      "slug": "london-city-airport",
      "alternative_names": [],
      "rank": "0",
      "timezone": "Europe/London",
      "city": {
        "id": "london_gb",
        "name": "London",
        "code": "LON",
        "slug": "london",
        "subdivision": "null",
        "autonomous_territory": "null",
        "country": {
          "id": "GB",
          "name": "United Kingdom",
          "slug": "united-kingdom",
          "code": "GB"
        },
        "region": {
          "id": "northern",
          "name": "Northern Europe",
          "slug": "northern-europe"
        },
        "continent": {
          "id": "europe",
          "name": "Europe",
          "slug": "europe",
          "code": "EU"
        }
      },
      "location": {
        "lon": "0.054167",
        "lat": "51.505"
      },
      "alternative_departure_points": [
        "id: LHR",
        "distance: 35.8",
        "duration: 1.4"
      ],
```

```
        "type": "airport"
      },
      {
        "id": "london_gb",
        "active": true,
        "name": "London",
        "slug": "london-united-kingdom",
        "code": "LON",
        "alternative_names": [],
        "rank": "0",
        "timezone": "Europe/London",
        "population": "7556900",
        "airports": "6",
        "stations": "0",
        "subdivision": "null",
        "autonomous_territory": "null",
        "country": {
          "id": "GB",
          "name": "United Kingdom",
          "slug": "united-kingdom",
          "code": "GB"
        },
        "region": {
          "id": "northern",
          "name": "Northern Europe",
          "slug": "northern-europe"
        },
        "continent": {
          "id": "europe",
          "name": "Europe",
          "slug": "europe",
          "code": "EU"
        },
        "location": {
          "lat": "51.507351",
          "lon": "-0.127758"
        },
        "type": "city"
      },
    ],
    "meta": {
      "locale": "",
      "status": ""
    }
}
```

400	Bad request format, missing request's params, invalid params' values

GET /locations/radius

Search by radius. This type of request supports either specification of location by coordinates (lat, lon) or by identifier (slug or id of location - term).

Parameters

Name	Description
apikey * `string` *(header)*	use the API key in the 'Data sources' section above.
lat `string` *(query)*	latitude of the centre point of the search. `40.7` is also acceptable. *Example* : 40.730610
lon `string` *(query)*	longitude of the centre point of the search. `-73.9` is also acceptable. *Example* : -73.935242
term `string` *(query)*	identifier of the location - slug or id (you cannot specify coordinates & term in the same request)
radius `string` *(query)*	the radius defaults to 250 km but can be changed *Example* : 250
locale `string` *(query)*	desired locale output - this is the language of the results. Should any other locale be used other than the specified locales, en-US is used. *Available values* : ar-AE, cs-CZ, da-DK, de-DE, el-GR, en-US, es-ES, fi-FI, fr-FR, hu-HU, is-IS, it-IT, iw-IL, ja-JP, ko-KR, lt-LT, ms-MY, nl-NL, no-NO, pl-PL, pt-BR, pt-PT, ro-RO, ru-RU, sk-SK, sr-RS, sv-SE, th-TH, tr-TR, uk-UA, zh-CN, zh-TW *Example* : en-US
location_types `string` *(query)*	desired location output, accepted values: `station`, `airport`, `bus_station`, `city`, `autonomous_territory`, `subdivision`, `country`, `region`, `continent`. To use more than one `location_types`, use multiple `&location_types=` *Example* : airport

limit `integer` *(query)*	default value = 20. Desired number of results in the output. *Example* : 20
sort `string` *(query)*	desired order of the output. For A->Z use 'sort=name', for Z->A use 'sort=-name'. Sorting by other fields, e.g. ‘sort=rank’, is possible.
active_only `boolean` *(query)*	default value = true. It displays all active locations. *Example* : true

Responses

Code	Description

200 **OK**

Media type: application/json

Example Value


```
{
  "locations": [
    {
      "id": "LCY",
      "int_id": "9625",
      "active": true,
      "code": "LCY",
      "name": "London City Airport",
      "slug": "london-city-airport",
      "alternative_names": [],
      "rank": "0",
      "timezone": "Europe/London",
      "city": {
        "id": "london_gb",
        "name": "London",
        "code": "LON",
        "slug": "london",
        "subdivision": "null",
        "autonomous_territory": "null",
        "country": {
          "id": "GB",
          "name": "United Kingdom",
          "slug": "united-kingdom",
          "code": "GB"
        },
        "region": {
          "id": "northern",
          "name": "Northern Europe",
          "slug": "northern-europe"
        },
        "continent": {
          "id": "europe",
          "name": "Europe",
          "slug": "europe",
          "code": "EU"
        }
      },
      "location": {
        "lon": "0.054167",
        "lat": "51.505"
      },
      "alternative_departure_points": [
        "id: LHR",
        "distance: 35.8",
        "duration: 1.4"
      ],
```

```
        "type": "airport"
      },
      {
        "id": "london_gb",
        "active": true,
        "name": "London",
        "slug": "london-united-kingdom",
        "code": "LON",
        "alternative_names": [],
        "rank": "0",
        "timezone": "Europe/London",
        "population": "7556900",
        "airports": "6",
        "stations": "0",
        "subdivision": "null",
        "autonomous_territory": "null",
        "country": {
          "id": "GB",
          "name": "United Kingdom",
          "slug": "united-kingdom",
          "code": "GB"
        },
        "region": {
          "id": "northern",
          "name": "Northern Europe",
          "slug": "northern-europe"
        },
        "continent": {
          "id": "europe",
          "name": "Europe",
          "slug": "europe",
          "code": "EU"
        },
        "location": {
          "lat": "51.507351",
          "lon": "-0.127758"
        },
        "type": "city"
      },
    ],
    "meta": {
      "locale": "",
      "status": ""
    }
  }
```


400	Bad request format, missing request's params, invalid params' values

404	Location specified by term (id or slug) does not exist

Search API

This document describes the endpoints that can be used to search flights. The flights can be aggregated by price per date and price per city.

- TLS protocol version 1.2 or later must be used.
- The correct date format is dd/mm/YYYY, e.g. 29/05/2021
- We do not support using body in GET requests.
- The responses can be gzipped, if request header accept-encoding:gzip, and need to be unpacked if response header is `Content-Encoding: gzip`
- Use a proper content type in the headers for all requests to Tequila API: `Content-Type: application/json`

GET /v2/search

A single flight search.

Parameters

Name	Description
apikey * **`string`** ***`(header)`***	use the API key in the 'Data sources' section above.
fly_from* *** ***`string` *`(query)`*	Kiwi api ID of the departure location. It accepts multiple values separated by comma, these values might be airport codes, city IDs, two letter country codes, metropolitan codes and radiuses as well as subdivision, region, autonomous_territory, continent and specials (Points of interest, such as Times Square). Some locations have the same code for airport and metropolis (city), e.g. DUS stands for metro code Duesseldorf, Moenchengladbach and Weeze as well as Duesseldorf airport. See the following examples: • 'fly_from=city:DUS' will match all airports in "DUS", "MGL" and "NRN" (all in the city of Duesseldorf) • 'fly_from=DUSf will do the same as the above • 'fly_from=airport:DUS' will only match airport "DUS"

Radius needs to be in form lat-lon-xkm. The number of decimal places for radius is limited to 6. E.g.`-23.24--47.86-500km` for places around Sao Paulo. 'LON' - checks every airport in London, 'LHR' - checks flights from London Heathrow, 'UK' - flights from the United Kingdom.
Example : FRA

fly_to
`string`
`(query)`

Kiwi api ID of the arrival destination. It accepts the same values in the same format as the 'fly_from' parameter
If you don't include any value you'll get results for all airports in the world.
Example : PRG

date_from *
`string`
`(query)`

search flights from this date (dd/mm/yyyy). Use parameters date_from and date_to as a date range for the flight departure.
Parameters 'date_from=01/05/2020' and 'date_to=30/05/2020' mean that the departure can be anytime between the specified dates.
Example : 05/12/2019

date_to *
`string`
`(query)`

search flights up to this date (dd/mm/yyyy)
Example : 25/12/2019

max_fly_duration
`integer`
`(query)`

max flight duration in hours, min value 0
Example : 20

flight_type
`string`
`(query)`

switch for oneway/round flights search - only oneway is supported. Will be deprecated in the near future.
Available values : round, oneway
Example : round

one_for_city
`integer`
`(query)`

It returns the cheapest flights to every city covered by the to parameter. E.g. if you set it to 1 and your search is from PRG to LON/BUD/NYC, you'll get 3 results: the cheapest PRG-LON, the cheapest PRG-BUD, and the cheapest PRG-NYC. `one_for_city` and `one_per_date` query parameters work only on one-way requests.
Example : 0

one_per_date
`integer`
`(query)`

returns the cheapest flights for one date. Can be 0 or not included, or one of these two params can be set to 1. `one_for_city` and `one_per_date` query parameters work only on one-way requests.
Example : 0

adults
integer
(query)

Used to specify the number of adults. Please note, that children are considered adults in our search engine. The default passengers' value is 1. The sum of adults, children and infants cannot be greater than 9.
Example : 1

children
integer
(query)

It is used to specify the number of children. The default value is 0. At the moment, children are considered adults in our search engine. We are working on improvements. The sum of adults, children and infants cannot be greater than 9.
Example : 1

infants
integer
(query)

It is used to specify number of infants. The default value is 0. The sum of adults, children and infants cannot be greater than 9.
Example : 1

partner_market
string
(query)

The market of a particular country from which the request originates. Use ISO 3166-1 alpha-2 to fill in the value.

curr
string
(query)

use this parameter to change the currency in the response
Available values : AED, AFN, ALL, AMD, ANG, AOA, ARS, AUD, AWG, AZN, BAM, BBD, BDT, BGN, BHD, BIF, BMD, BND, BOB, BRL, BSD, BTC, BTN, BWP, BYR, BZD, CAD, CDF, CHF, CLF, CLP, CNY, COP, CRC, CUC, CUP, CVE, CZK, DJF, DKK, DOP, DZD, EEK, EGP, ERN, ETB, EUR, FJD, FKP, GBP, GEL, GGP, GHS, GIP, GMD, GNF, GTQ, GYD, HKD, HNL, HRK, HTG, HUF, IDR, ILS, IMP, INR, IQD, IRR, ISK, JEP, JMD, JOD, JPY, KES, KGS, KHR, KMF, KPW, KRW, KWD, KYD, KZT, LAK, LBP, LKR, LRD, LSL, LTL, LVL, LYD, MAD, MDL, MGA, MKD, MMK, MNT, MOP, MRO, MTL, MUR, MVR, MWK, MXN, MYR, MZN, NAD, NGN, NIO, NOK, NPR, NZD, OMR, PAB, PEN, PGK, PHP, PKR, PLN, PYG, QAR, QUN, RON, RSD, RUB, RWF, SAR, SBD, SCR, SDG, SEK, SGD, SHP, SLL, SOS, SRD, STD, SVC, SYP, SZL, THB, TJS, TMT, TND, TOP, TRY, TTD, TWD, TZS, UAH, UGX, USD, UYU, UZS, VEF, VND, VUV, WST, XAF, XAG, XAU, XCD, XDR, XOF, XPD, XPF, XPT, YER, ZAR, ZMK, ZMW, ZWL
Example : EUR

locale
string
(query)

the language of city names in the response and also language of kiwi.com website to which deep_links lead
Available values : ae, ag, ar, at, au, be, bg, bh, br, by, ca, ca-fr, ch, cl, cn, co, ct, cz, da, de, dk, ec, ee, el, en, es, fi, fr, gb, gr, hk, hr, hu, id, ie, il, in, is, it, ja, jo, jp, ko, kr, kw, kz, lt, mx, my, nl, no, nz, om, pe, ph, pl, pt, qa, ro, rs, ru, sa, se, sg, sk, sr, sv, th, tr, tw, ua, uk, us, vn, za
Example : en

limit **`integer`** *`(query)`*	limit number of results, max is 200 *Example* : 30
sort **`string`** *`(query)`*	sorts the results by quality, price, date or duration. Price is the default value. *Available values* : price, duration, quality, date *Example* : price
asc **`integer`** *`(query)`*	can be set to 1 or 0, default is 1 - from cheapest flights to the most expensive *Example* : 1

Responses

Code	Description

200 **OK**

Media type: application/json

Example Value


```
{
  "search_id": "0e00b78e-91bb-449d-a1af-f5e626b3b602",
  "data": [
    {
      "id": "22ee0f6b491f000063ba729a_0",
      "duration": {
        "departure": 11220,
        "return": 0,
        "total": 11220
      },
      "flyFrom": "LGA",
      "cityFrom": "New York",
      "cityCodeFrom": "NYC",
      "countryFrom": {
        "code": "US",
        "name": "United States"
      },
      "flyTo": "MIA",
      "cityTo": "Miami",
      "cityCodeTo": "MIA",
      "countryTo": {
        "code": "US",
        "name": "United States"
      },
      "distance": 1770.31,
      "airlines": [
        "AA"
      ],
      "pnr_count": 1,
      "has_airport_change": false,
      "technical_stops": 0,
      "throw_away_ticketing": false,
      "hidden_city_ticketing": false,
      "price": 69,
      "bags_price": {
        "1": 36.96
      },
      "availability": {
        "seats": 7
      },
      "facilitated_booking_available": true,
      "conversion": {
        "EUR": 69
      },
      "quality": 142.86648200000002,
      "fare": {
```

```
      "adults": 34.5,
      "children": 34.5,
      "infants": 34.5
    },
    "price_dropdown": {
      "base_fare": 69,
      "fees": 0
    },
    "route": [
      {
        "fare_basis": "O7ALZNB3",
        "fare_category": "M",
        "fare_classes": "B",
        "fare_family": "",
        "return": 0,
        "bags_recheck_required": false,
        "vi_connection": false,
        "guarantee": false,
        "id": "22ee0f6b491f000063ba729a_0",
        "combination_id": "22ee0f6b491f000063ba729a",
        "cityTo": "Miami",
        "cityFrom": "New York",
        "cityCodeFrom": "NYC",
        "cityCodeTo": "MIA",
        "flyTo": "MIA",
        "flyFrom": "LGA",
        "airline": "AA",
        "operating_carrier": "AA",
        "equipment": "738",
        "flight_no": 1249,
        "vehicle_type": "aircraft",
        "operating_flight_no": "1249",
        "local_arrival": "2021-04-02T09:07:00.000Z",
        "utc_arrival": "2021-04-02T13:07:00.000Z",
        "local_departure": "2021-04-02T06:00:00.000Z",
        "utc_departure": "2021-04-02T10:00:00.000Z"
      }
    ],
    "local_arrival": "2021-04-02T09:07:00.000Z",
    "utc_arrival": "2021-04-02T13:07:00.000Z",
    "local_departure": "2021-04-02T06:00:00.000Z",
    "utc_departure": "2021-04-02T10:00:00.000Z"
  }
],
"currency": "EUR",
"fx_rate": 1,
"search_params": {
  "flyFrom_type": "airport",
  "to_type": "airport",
  "seats": {
```

```
        "passengers": 2,
        "adults": 1,
        "children": 1,
        "infants": 0
      }
    }
}
```

400	Bad request format, missing request's params, invalid params' values
422	"param": "fly_to"Not recognised location param "fly_to"