# Chapter 14 – An exploration of some V modules

Nov 30 2019: vpm contains 42 modules

## 14.1 Generating random data with Faker

<https://github.com/v-community/faker>

This module generates fake data. It is based on the Ruby gem [Faker library](https://github.com/faker-ruby/faker)

It uses the V flag module from vlib.

v install faker

error: Skipping module "faker", since it is missing name or url information. (it doesn’t appear to be published on vpm)

## 14.2 OWMW: an API wrapper for Open Weather Map.

OWMW is an acronym for Open Weather Map Wrapper. With OWMW you can easily gather data about the weather in the area. All you need is a (free) token from [OWM](<https://openweathermap.org/api>).

**A. Installing and using the module:**

The module Bowero.owmw is available via vpm and can be installed locally with:

C:\Users\CVO>v install Bowero.owmw

Installing module "Bowero.owmw" from https://github.com/Bowero/owmw to C:\Users\CVO\.vmodules/Bowero/owmw ...

(or the equivalent on Linux)

However fn init is pub, 🡪 compiler error

init -> start

building module doesn’t work on Windows, it does work on Linux:

v build module $HOME/owmw

* File $HOME/.vmodules/$HOME/owmw.o is created

v use\_owmw.v en use\_owmw werken dan, onder Windows werkt ook: v run use\_owmw.v

By installing the module, you have its source code. You can now start using it as described in the following section.

To use an installed module, you need to write a program that imports the module and calls the modules public API functions and methods (see § B.3). To figure these out, you need to consult the module’s documentation.

**B. Constructing the module:** (courtesy of ref. Bowero)

We’ll develop a module called *owmw*, to explore data from the open weather map (OWM).

You need to subscribe at the OWM site for a free account to the service. Then they will provide you with an API key token , which we need to access the site’s data.

B.1 Setting up the code:

First create a folder owmw, and there create a file *owmw.v* :

module owmw

const (

**base\_url** = 'https://api.openweathermap.org/data/2.5/'

)

pub fn **start**(token string, m int) **Weather** {

    mut metric := 'imperial'

    if (m == 1) {

        metric = 'metric'

    }

    return Weather {

        metric: metric

        token: token

    }

}

In it we declare the module. The OWM data is in JSON format, so in order to decode it, we need to import module json from vlib.

The base address for OWM is stored in a const base\_url.

The start function takes the token you need to provide to connect with OWM, and it also defines which metric system you want to use. The return value is of the Weather struct type:

struct **Weather** {

    token   string // this is the APPID: the API key you receive from OWM

    metric  string

}

To separate data definitions from functional code, we define all data structures we need in a file *structures.v*

More particularly, we need a struct *CityData*, which contains a number of other structs because the OWM JSON data is nested. These are all defined in the same file:

struct CityData {

    coord       CoordData

    weather     WeatherData

    base        string

    main        MainData

    visibility  int

    wind        WindData

    clouds      CloudsData

    dt          int

    sys         SysData

    id          int

    name        string

}

A value of this struct contains all weather data at the current time for a city with a given *name* or *id*.

Other data requests are also provided by OWM, but we use the city data as our example.

To test out our setup code, start writing a test program *use\_owmw.v*:

module main

**import owmw**

const (

    token = 'TOKEN' // TOKEN should be replaced with the API key you receive from OWM

)

fn main() {

    w := owmw.start(token, 1)

}

Compiling this gives us the warning that w is unused, but otherwise our module owmw compiles perfectly.

B.2 Asking for and decoding the data:

To get some data you need to send a request to the of OWM servers. For this we need the get method from module http.

To decode the received data, we need to import the json module.

We’ll gather all this code in a source file *functions.v*, which needs to do the imports:

**import (**

**http**

**json**

**)**

To see the exact format of our request, consult <https://openweathermap.org/current>

We see that the format is: api.openweathermap.org/data/2.5/weather?q={city name}

The first part is always the same, so we define this as the base\_url constant.

The url for the get request then takes the form:

url := '$base\_url/weather?q=$q&units=$w.metric&APPID=$w.token'

where q is the city name (for example: London, or Antwerp), w is the Weather struct value with kind of metric and the API key.

The function to get the data by city name takes this form:

pub fn (**w Weather**) city\_by\_name(q string) **CityData** {

    url := '$base\_url/weather?q=$q&units=$w.metric&APPID=$w.token'

    response := **w.city(url)**

    return response

}

It is in fact a method on the Weather struct. Then we call the city method with this url, which returns a CityData value as response:

fn (**w Weather**) **city**(url string) **CityData** {

    response := **http.get(url)** or {

        panic('Error getting Data: ' + err)

    }

    citydata := **json.decode(CityData, response.text)** or {

        panic(err)

    }

    return citydata

}

http.get returns an ?Response, so we need an or block when something goes wrong. The text property of the response is then decoded. Json.decode can also go wrong, so needs an or block also. The citydata returned is a complete CityData struct with the values for that city at the current time.

Now we can define all possible methods on CityData, such as the following for retrieving the temperature:

pub fn (c CityData) **temperature**() f32 {

    return **c.main.temp**

}

**C. Testing the service:**

In our test program *use\_owmw.v*, we import the owmw module and start the request for the service:

**import owmw**

const (

    token = '6f3e9a53f0be3733968163891bd22f5d'

)

fn main() {

    w := owmw.start(token, 1)

    city := **w.city\_by\_name('Antwerp')**

    println(city.temperature()) // 9.770000

}

The current temperature in Antwerp is 9.77 °C

## 14.3 Module categories

(See *v-modules.xlsx*)

*Web frameworks:*

|  |  |  |
| --- | --- | --- |
| 1) nedpals.vex | <https://github.com/nedpals/vex> | Web framework written on V inspired by Express and Sinatra. |
| 2) taojy123/**valval** | <https://github.com/taojy123/valval> | T[he fastest web framework in V language (vlang)](https://valval.cool/) |

<https://valval.cool/>