# Mars Met

*"A web application providing access to Mars weather"*

## 1) Introduction

Mars Met is a web application designed to provide easy access to Mars weather data. This data is collected in a daily basis by the REMS (Rover Environmental Monitoring System) of the MSL (Mars Science Laboratory), commonly known as "Curiosity". The data is uploaded to the web site of the "Centro de Astrobiología (CSIC-INTA)" in XML format.

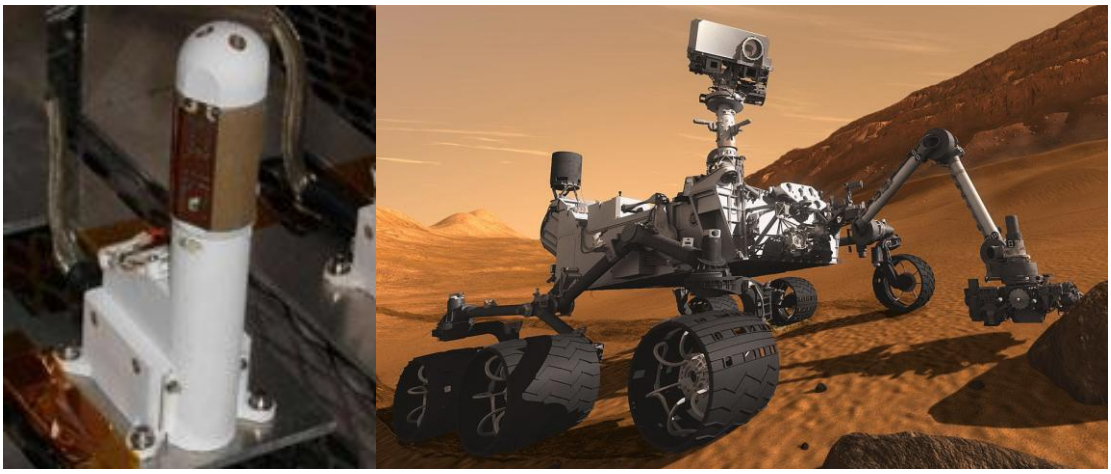

*Figure 1: The REMS instrument (left) and the Curiosity rover (right).*

This application relies basically on Django, a Python web development framework, PostgreSQL, an open source database engine, HighCharts, a JavaScript library for data plotting, and Bootstrap, a JavaScript and CSS collection for responsive design. Apart from this, Python's XML SAX and SQLAlchemymodules have been used for the data parsers.

## 2) Source data

As mentioned previously, the data is collected by the REMS subsystem of the Curiosity rover. This instrument can measure temperature, humidity, pressure, wind speed and wind direction. Unfortunately, the wind sensor was broken on the landing and the humidity data is also not well calibrated. Apart from this weather data, other information such as the sol (Martian solar day), the solar longitude of the planet, the sunrise and sunset times, etc. is provided.

This data is uploaded to the web site of "Centro de Astrobiología (CSIC-INTA)", the manufacturers of this subsystem, in XML documents called "weather reports". An example can be found below:

```
<weather_report>
      <title>
      RoverEnvironmentalMonitoringStationweatherreport
      </title>
      <link>https://cab.inta-csic.es/rems/</link>
      <sol>231</sol>
```

```
        <terrestrial_date>Apr 3, 2013 UTC</terrestrial_date>
        <magnitudes>
                <min_temp>-69.47</min_temp>
                <max_temp>3.05</max_temp>
                <pressure>889.18</pressure>
                <pressure_string>Higher</pressure_string>
                <abs_humidity>--</abs_humidity>
                <wind_speed>2</wind_speed>
                <wind_direction>E</wind_direction>
                <atmo_opacity>Sunny</atmo_opacity>
                <season>Month 10</season>
                <ls>293.8</ls>
                <sunrise>6 am</sunrise>
                <sunset>5 pm</sunset>
        </magnitudes>
</weather_report>
```

Apart from this daily reports, which are updated when a new report comes, there is an historic XML file in the web site "Mars Weather" of Ashima Research. This is called "climate report" and stores all the weather reports.

The main problem of this data is its low accessibility. A normal person or even a scientist needs an application that reads these XML files and gives him the information. The existing applications provide the data in very static graphs, which may not be really useful.

### 3) Data parsing

To make data more accessible a database has been used. This database uses the PorstgreSQL open source engine. This way, the applications can access data easily with simple SQL queries. This is also a good point if there is a need to create an API.

Two tables have been created in the database, one for the "reports", which stores the "sol" and the corresponding "terrestrial date", and one for the "magnitudes", which stores the weather data and is indexed by the "sol".

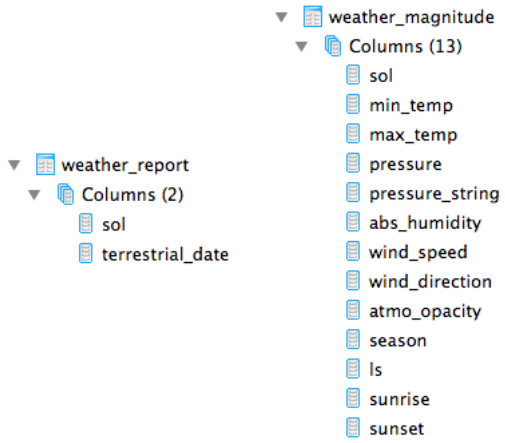


*Figure 2: Database tables and columns. Report table (left) and Magnitude table (right).*

When the tables are created, the data has to be parsed from the XML documents. For this porpoise, a data parser has been developed with Python, based on the XML

SAX module to read the source files and on the SQLAlchemy module to feed the database.

### *3.1) Challenges – Terrestrial Date*

During the data parser coding process, the team realized that the "terrestrial date" field of the source XML had several formats. This made quite difficult the parsing process, since this field had to be checked many times.

To overcome this problem, a scientific calculus to obtain the terrestrial date based on the "sol" field was developed. Knowing Curiosity's mission day zero ($S_0$) and the length of a Martian solar day ($\Delta S$), we can obtain the desired result ($S_i$), with an accuracy of seconds:

$$S_o = 2012 - 08 - 05\ T\ 13{:}49{:}00$$

$$\Delta S = 24{:}39{:}35.244$$

$$S_i = S_0 + i \cdot \Delta S$$

## 4) Web Application

The web application was developed with the Django web framework. This is a fully Python framework based on the MVC architecture. It provides a fast and simple set of tools to develop a web application that relies on a database.

### *4.1) Models*

The models were set according to the database templates. One important feature for the administrators is that they can modify data easily from Django's admin interface.

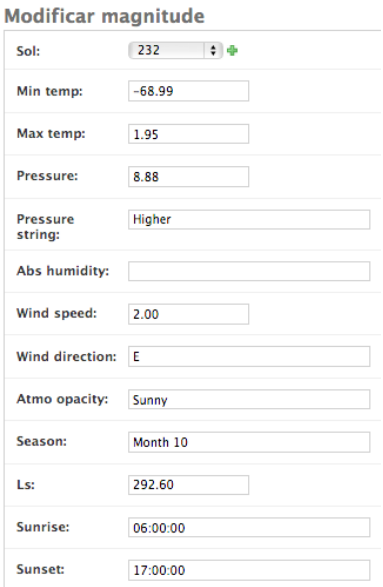


*Figure 3. Example of a "magnitude" modification from the admin interface.*

### *4.2) Views*

The views are quite simple. They just provide data to the HTML and JavaScript code in form of QuerySets. This is done this way, because most of the plotting work is performed with JavaScript, using the HighCharts API. In the beginning, the charts were designed in the views using Chartit, a Python interface to the HighCharts API. However, this doesn't provide all the features that the web app demanded.

### *4.3) Challenges – Earth location*

One of the columns of the magnitude table stores the Martian solar longitude, this is, the position of Mars in its orbit. We thought that it would be nice to have also this parameter but for the Earth. This way, we could plot the relative location of both planets for each day.

For this porpoise a scientific calculus based on Kepler'ssecond law was performed. This new was calculated in the models script, since it can be then provided as a property of the report model, as if it was stored in the database.

The calculus is done in the following way. First, as input data we need the time interval (Δt) between the current day and the January 2nd of the same year, since the perihelion is reached this day.

$$\Delta t = T_{today} - T_{Jan\ 2^{nd}}\ (\text{seconds})$$

Knowing the angular velocity of the Earth (n), we can compute the angular sector swept (M) during this interval:

$$n = \frac{2\pi}{365.256 \cdot 86400}\ (radians/second)$$

$$M = n \cdot \Delta t\ (radians)$$

This initial position (M) is calculated considering a circular movement. However, the real movement is elliptical, with a known eccentricity (e). To correct this problem, an iterative process is applied. The elliptical position ($E_e$) is calculated using the eccentricity and corrected in each step, until a minimum error is achieved:

$$E_c = M$$
$$\downarrow$$
$$E_e = M + e \cdot \sin E_c$$
$$\downarrow$$
$$¿\, E_e \approx E_c ?$$

Finally, when this optimal position with minimum deviation is found, the angle of the solar longitude can be computed with the following formula:

$$\theta = 2 \cdot \tan^{-1}\left[\sqrt{\frac{1+e}{1-e}} \cdot \tan\left(\frac{E_e}{2}\right)\right] (radians)$$

For a proper representation of this data, the cosine and sine of this angle is calculated and they are multiplied by the mean distance of the planet to the Sun in Astronomical Units:

$$d_x = d_{UA} \cdot \cos\theta$$

$$d_y = d_{UA} \cdot \sin\theta$$

## 5) Conclussions

This is our first experience in a challenge like that, so when we started programming we just wanted to participate in a so huge event. Working in team with some work buddies and seeing how other teams worked was very interesting, not only in our town but also in other countries.

Programming was also interesting, since we are students and we are used to work in projects without clear applications and not so imaginative. Working on this challenge broke our daily routine, and gave us a chance to enjoy programming with other people.