Mapping Museums: Web Application documentation.

Nick Larsson, Val Katerinchuk, Alex Poulovassilis *

May 21, 2020

Abstract

This document describes the web application developed by the Mapping Museums project, its architecture and components as well as installation and extension. It also documents links to the data loading process and how they work together.[1]

*Department of Computer Science and Information Systems, Birkbeck, University of London. nick.larsson@ymail.com, valery.katerinchuk@gmail.com, ap@dcs.bbk.ac.uk

[1]The Mapping Museums project is funded by the UK Arts and Humanities Research Council (AHRC), 2016-2020, Ref AH/N007042/1. Details of the project can be viewed at `www.mappingmuseums.org`

Contents

1 Introduction

The project has been realised using the Python Flask framework for the backend services with the addition of some Flask APIs to the views. The front end is composed of Flask views using the Bootstrap framework, Leaflet for geolocation data and Bokeh as a plotting package. Mapping is achieved using the OpenStreetMaps API, Automatic Street View generation is created using JavaScript and Instant Street View services, and visualisation screenshots created using html2canvas. Javascript is used in the web pages for interaction, and the Flask views load the data as Javascript structures. The web application has the usual structure for a Flask framework application as shown in Figure 1.

Note: This application is a research prototype that was developed under tight time and resource constraints to deliver the necessary functionalities for investigating the Mapping Museums project's research questions. Little code refactoring has taken place. It may or may not be useful beyond its stated scope and no guarantees of accuracy are given or implied. The code is free to use under the GNU GPL3 license. Details of our design and development methodology can be found in [1].

2 Software and packages

These are listed in Tables 1 and 2.

Software	**Use**
Virtuoso	Used as the backend data store
RDF/RFS	Used to encode the data and model for storage in the data store
Python	Used for programming the data quality checks and transformations
Bootstrap	The web application framework for creating web pages
Bokeh	Used for programming the visualisations

Table 1: Back-end softwares used by the project

We gratefully acknowledge the use of the following software under the terms of the specified licences:

Software	Use
Bokeh	Plotting package for the Visualisations
Leaflet	Geolocation library to show museums on a map; used in Browse and Search
Bootstrap	Web application framework for web pages
awesomplete	Javascript package for predictive text. Used in the Search over Admin Areas
MarkerCluster	Addition to Leaflet to show clusters on a map
OpenStreetMaps	Used for generating the maps in Browse and Search
html2canvas	Used for imaging the graphs in Visualise
nouislider	Javascript package implementing a slider. Used with Leaflet
treecss	Used to implement all menus as trees in CSS for performance
rtree	Tree data structure implementation used to hold map data and access the markers efficiently
tether	A JavaScript library for efficiently making an absolutely positioned element stay next to another element on the page.
wNumb	JavaScript Number and Money formatting.

Table 2: Front-end softwares used by the project

Google Street View https://www.google.com/permissions/geoguidelines/
html2canvas - https://github.com/niklasvh/html2canvas/blob/master/LICENSE
Virtuoso http://vos.openlinksw.com/owiki/wiki/VOS/VOSLicense

3 Requirements and installation

The Python library requirements are as follows:

```
alabaster==0.7.11
aniso8601==2.0.0
appdirs==1.4.0
Babel==2.6.0
backports-abc==0.5
backports.shutil-get-terminal-size==1.0.0
beautifulsoup4==4.5.3
bokeh==0.12.14
bs4==0.0.1
certifi==2018.1.18
chardet==3.0.4
click==6.7
cycler==0.10.0
decorator==4.0.11
django-multiforloop==0.2.1
docutils==0.14
dominate==2.3.1
enum34==1.1.6
et-xmlfile==1.0.1
Flask==0.12
Flask-Bootstrap==3.3.7.1
Flask-Compress==1.4.0
Flask-Moment==0.5.1
Flask-RESTful==0.3.6
Flask-Script==2.0.5
Flask-WTF==0.14.2
flexx==0.4.1
functools32==3.2.3.post2
funkload==1.17.1
futures==3.2.0
fuzzywuzzy==0.14.0
idna==2.7
imagesize==1.0.0
```

```
ipython==5.5.0
ipython-genutils==0.2.0
isodate==0.5.4
itsdangerous==0.24
jdcal==1.3
Jinja2==2.9.5
lxml==3.7.2
MarkupSafe==0.23
matplotlib==2.0.0
networkx==1.11
numpy==1.12.0
openpyxl==2.4.2
packaging==16.8
pandas==0.19.2
pathlib2==2.3.0
Pattern==2.6
pexpect==4.2.1
pickleshare==0.7.4
Pillow==5.0.0
pkg-resources==0.0.0
prompt-toolkit==1.0.15
ptyprocess==0.5.2
pyexpander==1.7.0
Pygments==2.2.0
pygraphml==2.2
pyparsing==2.1.10
python-dateutil==2.6.0
pytz==2016.10
rdflib==4.2.2
requests==2.19.1
scandir==1.6
simplegeneric==0.8.1
singledispatch==3.4.0.3
six==1.10.0
SPARQLWrapper==1.8.0
subprocess32==3.2.7
traitlets==4.3.2
typing==3.6.4
urllib3==1.23
visitor==0.1.3
wcwidth==0.1.7
```

```
webunit==1.3.10
Werkzeug==0.11.15
WTForms==2.1
xlrd==1.0.0
```

The front-end softwares are delivered with the project in the Flask structure in the JS directory. The back end needs to have a SPARQL endpoint to issue queries against which is defined in the file app/searchapplication.cfg:

```
URLREWRITEPATTERN=193.61.36.71/ # URI to URL transformation patterns
SPARQLENDPOINT=http://193.61.36.21:8890/sparql #
DEFAULTGRAPH=http://bbk.ac.uk/MuseumMapProject/graph/v10 # Data graph
GEOADMINGRAPH=http://bbk.ac.uk/MuseumMapProject/graph/ukadmin # ONS data
DEV_MODE=F # Run in dev or prod mode
```

4 Application architecture

4.1 Logical architecture

The logical architecture is a classic web application with a view server (Flask) connecting to a database (Virtuoso) with web client browsers using Javascript as seen in Figure 1. The Flask views correspond with the tabs in the web page: Browse, Search and Visualise. To extend the application, a new view would be created to accommodate the new functionality. There is also an API service which serves the current data version — this is v10, which includes also data about the Deprivation Index[2] and Area Classification[3] relating to the administrative area of each museum, as well as names from the Office for National Statistics (ONS) dataset for UK Administrative Areas[4].

4.2 Component architecture

The server side application is a structured as Flask project application with a blueprint and an API service, see Figure 2 and Figure 3 for the files.

[2]https://www.gov.uk/government/statistics/english-indices-of-deprivation-2015, https://gov.wales/statistics-and-research/welsh-index-multiple-deprivation/?lang=en, http://simd.scot/2016/, https://www.nisra.gov.uk/statistics/deprivation/northern-ireland-multiple-deprivation-measure-2017-nimdm2017

[3]https://www.ons.gov.uk/methodology/geography/geographicalproducts/areaclassifications/2011areaclassifications/datasets

[4]https://data.gov.uk/dataset/7709b64e-369f-41f4-96ce-1f05efde9834/national-statistics-postcode-lookup-august-2017

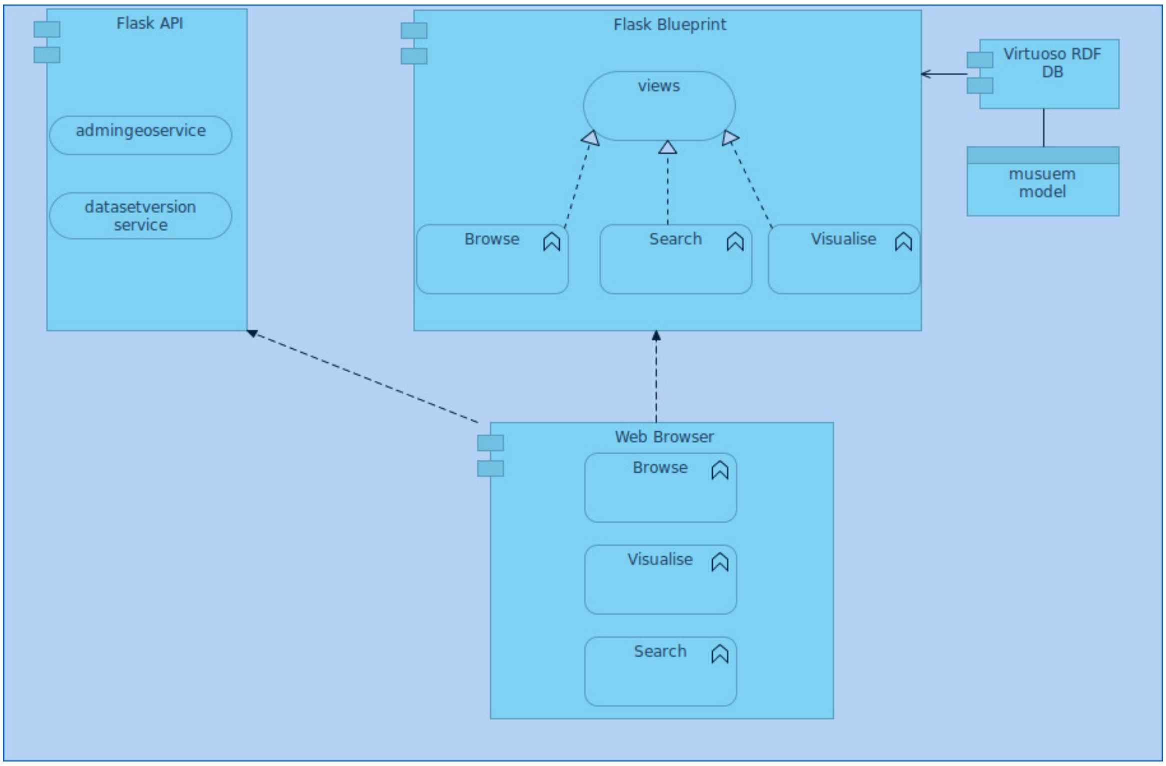

Figure 1: Flask application structure

The modules are divided into view implementations (search, browse etc), helper modules (second and third columns) and implementation of the datatypes.

The **models** module executes first in the application and initialises the following:

- Predicates in the data model
- Datatypes (classes) in the model
- Search menus and query configuration
- Columns to show in single museum view (nakedview)
- Caches all lists in data model
- Reads all JSON files for Leaflet

Each of the view implementations also has extensive initialisation sections to build the large menus with thousands of options. This is done lazily on the first call to the view and depending on the DEV flag (see Section 3) either builds the models (=T) or loads them from a file-based serialisation (=F).

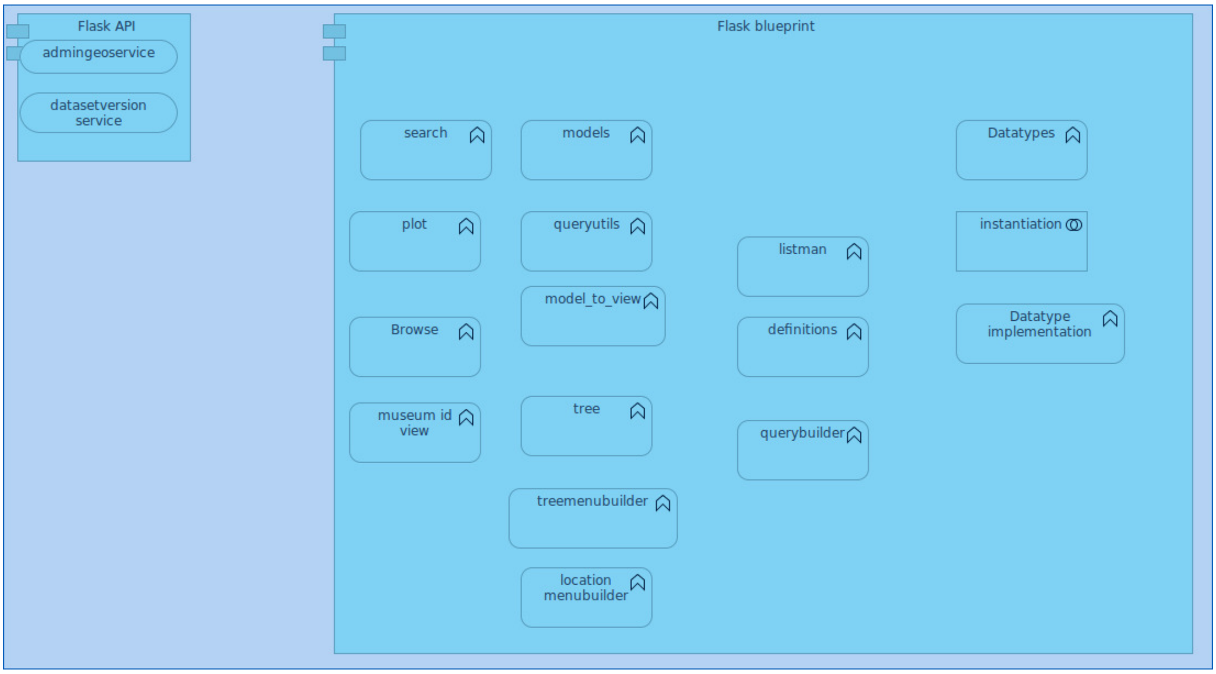

Figure 2: Flask application component structure

The python module and class structure can be seen in Figure 4. It separates all visualisation implementations in the boksplots directory, datatype implementations, and implementations of tree libraries used for the menu building.

The **boksplots** directory contains a file for each type of plot with self-explanatory names. The category plots handle category data such as classifications while the time plots handle time events. The specification of the statistical computations underpinning the visualisations can be downloaded from `http://museweb.dcs.bbk.ac.uk/software`.

The **datatypes** directory contains the implementation of the datatypes in the system that are not classes from the main spreadsheet. Thus each datatype refers to an individual spreadsheet with its own Extract-Transformation-Load (ETL) process. The datatype implementation requires an interface to be implemented as shown in Table 3. Full details can be found in the source code.

The **treelib** module contains the implementation of the tree data structure. The individual files have functions as shown in Table 4.

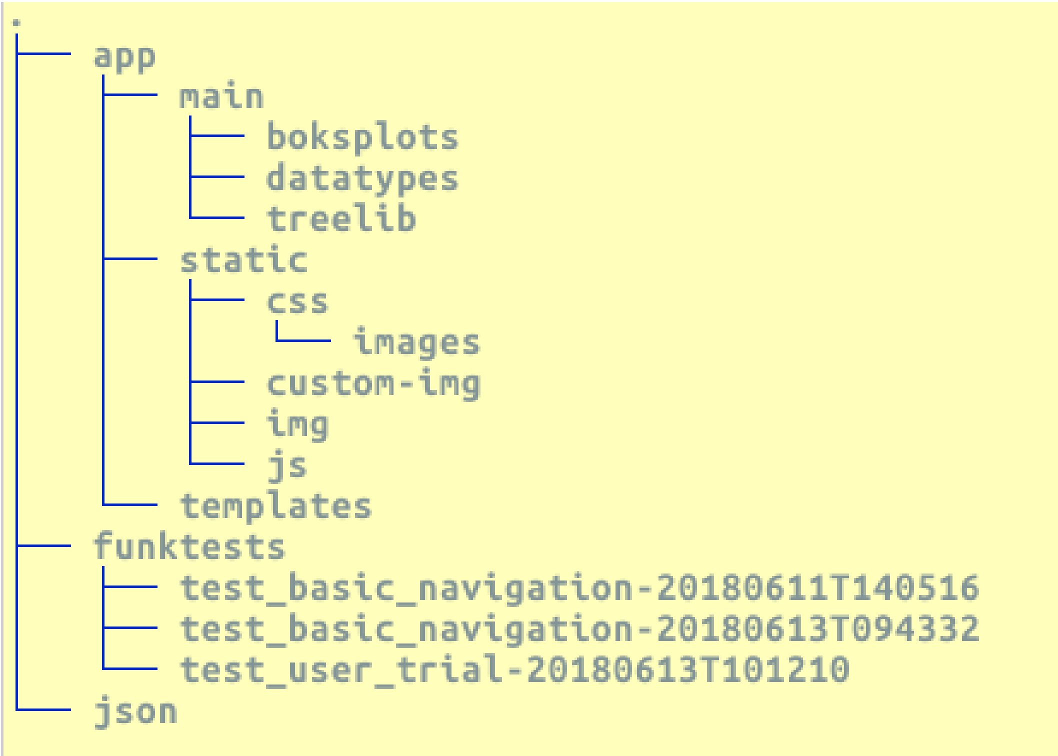

Figure 3: Flask application file structure

4.3 Physical architecture

The physical architecture is a traditional database-centric one, reflected by the logical architecture where the database is on one node (musedb) and the application server on another (museweb). Museweb is on the internet with port 80 and musedb is only accessible on the DCS intranet, see Figure 7.

4.3.1 Database/dataset and model setup

The web application is expecting a setup file with the SPARQL endpoint and the graphs to query as described in Section 3. The **models** initialisation queries the database for model information:

- Values of **List** classes.
  For example for the Accreditation we expect an RDFList named bbkmm:AccreditationList containing all the possible values : *("Accredited" "Unaccredited")*
- All predicates for the ETL subtask.
  The RDFList is named bbkmm:PredicateList_{$ETLsubtask}, e.g. main-

```
.
├── app
│   ├── __init__.py
│   ├── main
│   │   ├── admingeoservice.py
│   │   ├── apputils.py
│   │   ├── boksplots
│   │   │   ├── bokehcategorybar.py
│   │   │   ├── bokehcategorypie.py
│   │   │   ├── bokehheatmap.py
│   │   │   ├── bokehtimeandcategory.py
│   │   │   ├── bokehtime.py
│   │   │   ├── bokehtree.py
│   │   │   ├── bokutils.py
│   │   │   ├── custompie.py
│   │   │   └── __init__.py
│   │   ├── boks.py
│   │   ├── chloro.py
│   │   ├── Configuration.py
│   │   ├── datasetversion.py
│   │   ├── datasetversionservice.py
│   │   ├── datatypes
│   │   │   ├── Admin_Area.py
│   │   │   ├── Governance_Change.py
│   │   │   ├── __init__.py
│   │   │   └── Visitor_Numbers_Data.py
│   │   ├── definitions.py
│   │   ├── errors.py
│   │   ├── __init__.py
│   │   ├── listman.py
│   │   ├── mapchart.py
│   │   ├── models.py
│   │   ├── model_to_view.py
│   │   ├── nakedid.py
│   │   ├── prova_england_Enumbers.py
│   │   ├── prova_location_Enumbers.py
│   │   ├── PTreeNode.py
│   │   ├── search.py
│   │   ├── showmuseumtypes.py
│   │   ├── treelib
│   │   │   ├── exceptions.py
│   │   │   ├── __init__.py
│   │   │   ├── node.py
│   │   │   ├── plugins.py
│   │   │   └── tree.py
│   │   ├── tree.py
│   │   └── views.py
│   ├── static
│   │   ├── css
│   │   │   └── images
│   │   ├── custom-img
│   │   ├── img
│   │   └── js
│   └── templates
├── config.py
├── fab_dcs.py
├── fab_deploy.py
├── funktests
│   ├── test_basic_navigation-20180611T140516
│   ├── test_basic_navigation-20180613T094332
│   ├── test_BasicNavigation.py
│   ├── test_user_trial-20180613T101210
│   └── test_UserTrial.py
├── json
└── manage.py
```

Figure 4: Python modules and classes

Interface	Use
getMatchFilter(self,rcount,match, condition)	Returns SPARQL for a match condition filter on the data type
getCompareFilter(self,rcount,match, condition)	Returns SPARQL for a condition filter on the data type
getQuery(self,col,rcount,matchstring, condition,matchcolumn)	Returns SPARQL for a query on the datatype without filter
getSearchType(self)	Returns type for the datatype to appear in search menu
getGUIConditions(self)	Returns the list of select conditions for the comaparator search menu
getWidget(self)	Returns html code for search menu
getWidgetCode(self)	Returns JS code associated with the HTML for the datatype

Table 3: Data type interface

sheet and bbkmm:PredicateList_mainsheet. The content could be : *("bbkmm:hasName_of_museum" "bbkmm:hasACE_size_source")*

- All classes and their datatypes.
  The RDFList is named bbkmm:DataTypeList_{$ETLsubtask}, e.g. mainsheet and bbkmm:DataTypeList_mainsheet. The content could be : *("defSize#ListType" "defRangeYear_closed#RangeType")*

4.3.2 Initialisation of models

The graph is queried for model information as shown in the previous section from the **models** module using methods in the **apputils** and **listman** modules. The information is stored in the **definitions** module for access from the whole web application.

4.3.3 ONS data integration

The ONS data is stored in a CSV file named **NSPL_AUG_2017_UK.csv** which contains the 2017 postcode data set. Prior to installation please download this file from `http://museweb.dcs.bbk.ac.uk/static/pdf/NSPL_AUG_2017_UK.csv` and move it to the museumflask/json directory. The initialisation of this ONS data consists of building dictionaries that allow us to

File	Use
admingeoservice.py	Service for predictive text for Search
apputils.py	Implements query engine
boks.py	Routing of the many visualisation alternatives to the correct method
chloro.py	Chloropleth for various regions, not used at the moment
Configuration.py	Configuration view, not used at the moment
datasetversionservice.py	Returns the data set version currently in use
definitions.py	Definition of variables used globally in the application
errors.py	Error pages
listman.py	List management for data model helpers
mapchart.py	Map view
models.py	Initialisation of models in the application
model_to_view.py	Conversions between model and views
nakedid.py	View of one museum without context
PTreeNode.py	Menu tree node implementation
search.py	Search view implementation
showmuseumtypes.py	Browse view implementation
tree.py	Simple tree implementation
views.py	Routing of all urls to the correct implementation

Table 4: Application files

traverse the ONS administrative area hierarchy from country down to local government. This code can be found in the **apputils** module together with initialisation of ONS Ecodes and names.

4.4 Datafiles used

All initialisation data is stored in the **JSON** directory. It contains a number of geojson files used for presenting maps with leaflet. These files are loaded from the **models** module to the **definitions** module to be accessed from the views. The names are self explanatory and contain lat/long information for all museums to be shown by Leaflet. In addition it contains the definition files shown in Table 5

File	**Use**
boksgreeting.html	This file ends up as landing page for the Visualisation tab
county.csv	List of all ONS counties
DEFAULT_BROWSE_COLUMNS.txt	First part of the menu tree filters in Browse
DEFAULT_BROWSE_COLUMNS2.txt	Second part of the menu tree filters in Browse
DEFAULT_SEARCH_FILTER_COLUMNS.txt	The Search filters
DEFAULT_SEARCH_SHOW_COLUMNS.txt	The default items to show for Search
DEFAULT_VIEW_ALL_COLUMNS.txt	List of items to show when ALL is chosen in Search
DEFAULT_TABLE_PLOT_COLUMNS.txt	The default selections for the Table plots in Visualise
distr.csv	District id to name dictionary
LocalAuthMap.csv	Local auth id to name dictionary
NSPL_AUG_2017_UK.csv	ONS postcode dataset as one csv file

Table 5: Definition files

4.5 Data type handling and links to ETL

The web application expects some naming conventions, as shown in Section 4.3.1. The implemented abstract data types are referred to as follows, taken from module **definitions**:

```
##  datatype naming definitions
HASNAME   = "has"
DEFRANGE  = "defRange"
DEFCLASS  = "defClass"
DEFNAME   = "def"
```

```
RANGENAME = "Range"
LISTNAME  = "List"

##  datatypes definitions for the abstract types
DEFINED_TYPES ="HierType","ListType","RangeType"
```

These conventions allow for calculating the correct predicate and class names, and if the class name starts with def an abstract type. The types themselves are defined in csv file input to the ETL processs.

4.6 Query engine implementation

The major challenge in the query engine implemented in the **apputils** module has been to keep filters and query variables in synch. For each property to query, a new SPARQL id needs to be generated that is unique. Therefore a variable **rcount** is used keep the current variable id across query and filter generation. There is also a **coltoargdict** dictionary variable that links a spreadsheet column with a SPARQL variable. The best way to understand how this works is to look at a data type implementation of one of the interfaces in Table 3.

5 Data models

5.1 XSD data type support

The web application supports the XSD data types necessary to represent the musuem data. They are as listed in module **definitions**:

```
##  xmldatatypes definitions for the xsd types
XML_TYPES=['string','integer','positiveInteger','date','boolean','decimal'];
XML_TYPES_WITH_PREFIX=['xsd:string','xsd:integer','xsd:positiveInteger',
                       'xsd:date','xsd:boolean','xsd:decimal'];
XML_TYPES_PREFIX="xsd:"
```

5.2 Abstract data models from ETL

In addition, the data model contains some abstract data types to handle date ranges, lists (bag of words) and hierarchies, see Table 6. These datatypes are understood by the web application and this speeds up the modeling of data which naturally falls in to these categories. The abstract types are defined in module **definitions**:

```
##  datatypes definitions for the abstract types
DEFINED_TYPES = "HierType","ListType","RangeType"
```

Datatype Model	Use
bbkmm:NameList	Used to create a bag of words from a column in the source data spreadsheet
range:datatype	Used to create a time range to hold start and end dates. The range data is typically positiveInteger or date
hier:NamedHierarchy	Used to create a subclass hierarchy from a column in the source data spreasheet

Table 6: Complex datatypes

6 Visualisations

Several packages were tried before settling on Bokeh as the package with the cleanest look and support for the functions needed. It suffers as all web based plotting from the need to implement event handling and actions in the front-end language (JS) rather than the back-end language (python) where all the calculations occur.

6.1 Bokeh package, routing and components

The back end needs the Bokeh library installed as shown in the Requirements section. The number of possible plots and combinations is large, in the thousands, so routing of plot requests is an issue. The routing is all handled by the Boks module which has the responsibility to build the menu tree that allows for different plots to be called and the handling of these. The ONS dictionaries are used again for the *Location* menus. Individual plots are implemented in the boksplots directory. This includes some types such as tree plots which are not currently active but could be considered as an extension to the system. The ONS model of Administrative Areas that we use can be viewed in the schema diagrams accessible at `http://museweb.dcs.bbk.ac.uk/data`.

6.2 Computations

The statistical computations that underpin the numbers of museums generated for each visualisation are specified in a document accessible at `http://museweb.dcs.bbk.ac.uk/software`. The code in the boksplots implementations mirrors this specification to ensure the correctness of the computations.

7 Maps

Both Browse and Search contain tabs for viewing museum locations on a map. The maps are supported by the Leaflet package and the OpenStreetMaps server.

7.1 GeoJSON files

The overlays on the maps come from *geoJSON* files all initialised from the **JSON** directory in the **models** module.

7.2 Providing Leaflet with data

Data for the maps is either provided as a Javascript array (search, browse) or as a properties in a geoJSON file (map). The geoJSON featureset is illustrated below:

```
"type": "Feature", "properties":
     {"objectid": 1,
      "musfreq": 37,
      "cty15cd": "E10000002",
      "bname": "Buckinghamshire",
      "st_areashape": 1564949146.6724994,
      "st_lengthshape": 361852.5309305974}},
```

The frequency enables the showing of a chloropleth in Leaflet - not currently used.

8 Front end

The front end follows the *Flask* framework with the *Javascript* code provided by files in the **js** directory and *CSS* files in the **css** directory, see Figures 5 and 6. The application uses Jinja templates to deliver data into the page from the back end views. All views derive from the base template, base.html.

8.1 Document structure

Table 7 gives an explanation of the use for each template. It follows roughly the back end view naming as expected.

File	Use
base.html	The bootstrap base from which all templates are derived
boksplot.html	Landing page for Visualisations
browseproperties.html	Landing page for Browse
index.html	Home page
message.html	General failure message template
nakedchloro.html	Chlorograph page without menus
nakedid.html	Individual museum page without menus
nakedmap.html	Individual map page without menus
nakedresults.html	Results page without menus
search.html	Search page menus
searchmessage.html	Failure message
searchResults.html	Search results in subframe complementing the menus
showpage.html	Shows a static page with bootstrap decorations
visualisation.html	Visualisations page

Table 7: Static view templates

8.2 Menu system

The menus use variations on the tree structure implemented in *CSS* in order to accomodate many nodes without performance problems resulting from Javascript execution. The **treecss.css** renders the tree generated by the **tree.py** implementation. The python implementation enables you to build a tree programmatically of any size and complexity and output this as an HTML list structure to be rendered by CSS. This is done by all views (search, browse, visualise) to generate the tree structure that is the menu.

9 Deployment of web application

The web application is deployed on an *Apache2* installation using the *WSGI* module for *python Flask*. Automatic deployment is done with *Fabric* and is not shown here as it is specific to Birkbeck's intranet.

10 Authentication in the web application

Authentication can be easily switched on with the help of *Apache Basicauth* for *WSGI* as shown in the configuration below:

```
<VirtualHost *>
    ServerName museweb.dcs.bbk.ac.uk
    WSGIScriptAlias / /var/www/museumflask/museumflask.wsgi
    WSGIDaemonProcess museumflask
    <Directory /var/www/museumflask>
       WSGIProcessGroup museumflask
       WSGIApplicationGroup %{GLOBAL}
        Order deny,allow
        AuthType Basic
        AuthName "MuseumAuth"
        AuthBasicProvider wsgi
       WSGIAuthUserScript /usr/local/wsgi/scripts/auth.wsgi
       Require valid-user
    </Directory>
</VirtualHost>
```

References

[1] A. Poulovassilis, N. Larsson, F. Candlin, J. Larkin, and A. Ballatore. Creating a Knowledge Base to research the history of UK Museums through Rapid Application Development. *ACM Journal on Computing and Cultural Heritage*, 12(4):1–27, 2020.

```
.
├── app
│   ├── main
│   │   ├── boksplots
│   │   ├── datatypes
│   │   └── treelib
│   ├── static
│   │   ├── css
│   │   │   └── images
│   │   ├── custom-img
│   │   ├── img
│   │   └── js
│   │       ├── alert.js
│   │       ├── awesomplete.js
│   │       ├── bootstrap.js
│   │       ├── bootstrap-multiselect-collapsible-groups.js
│   │       ├── bootstrap-multiselect.js
│   │       ├── button.js
│   │       ├── carousel.js
│   │       ├── collapse.js
│   │       ├── dropdown.js
│   │       ├── jquery.js
│   │       ├── jquery_min_1_11_1.js
│   │       ├── jquery.range-control.js
│   │       ├── karma.conf.js
│   │       ├── leaflet-color-markers.js
│   │       ├── leaflet.js
│   │       ├── leaflet.layerindex.js
│   │       ├── Leaflet.MakiMarkers.js
│   │       ├── leaflet.markercluster.js
│   │       ├── modal.js
│   │       ├── nouislider.js
│   │       ├── popover.js
│   │       ├── rtree.js
│   │       ├── scrollspy.js
│   │       ├── tab.js
│   │       ├── tether.js
│   │       ├── tooltip.js
│   │       ├── util.js
│   │       └── wNumb.js
│   └── templates
├── funktests
│   ├── test_basic_navigation-20180611T140516
│   ├── test_basic_navigation-20180613T094332
│   └── test_user_trial-20180613T101210
└── json
```

Figure 5: JS packages

```
.
├── app
│   ├── main
│   │   ├── boksplots
│   │   ├── datatypes
│   │   └── treelib
│   ├── static
│   │   ├── css
│   │   │   ├── awesomplete.base.css
│   │   │   ├── awesomplete.css
│   │   │   ├── awesomplete.theme.css
│   │   │   ├── bootstrap.css
│   │   │   ├── bootstrap.min.css
│   │   │   ├── bootstrap-theme.css
│   │   │   ├── bootstrap-theme.min.css
│   │   │   ├── images
│   │   │   ├── leaflet.css
│   │   │   ├── MarkerCluster.css
│   │   │   ├── MarkerCluster.Default.css
│   │   │   ├── nouislider.css
│   │   │   ├── style.css
│   │   │   ├── tether.css
│   │   │   ├── tether.min.css
│   │   │   ├── tether-theme-arrows.css
│   │   │   ├── tether-theme-arrows-dark.css
│   │   │   ├── tether-theme-arrows-dark.min.css
│   │   │   ├── tether-theme-arrows.min.css
│   │   │   ├── tether-theme-basic.css
│   │   │   ├── tether-theme-basic.min.css
│   │   │   └── treecss.css
│   │   ├── custom-img
│   │   ├── img
│   │   └── js
│   └── templates
├── funktests
│   ├── test_basic_navigation-20180611T140516
│   │   └── funkload.css
│   ├── test_basic_navigation-20180613T094332
│   │   └── funkload.css
│   └── test_user_trial-20180613T101210
│       └── funkload.css
└── json
```

Figure 6: CSS packages

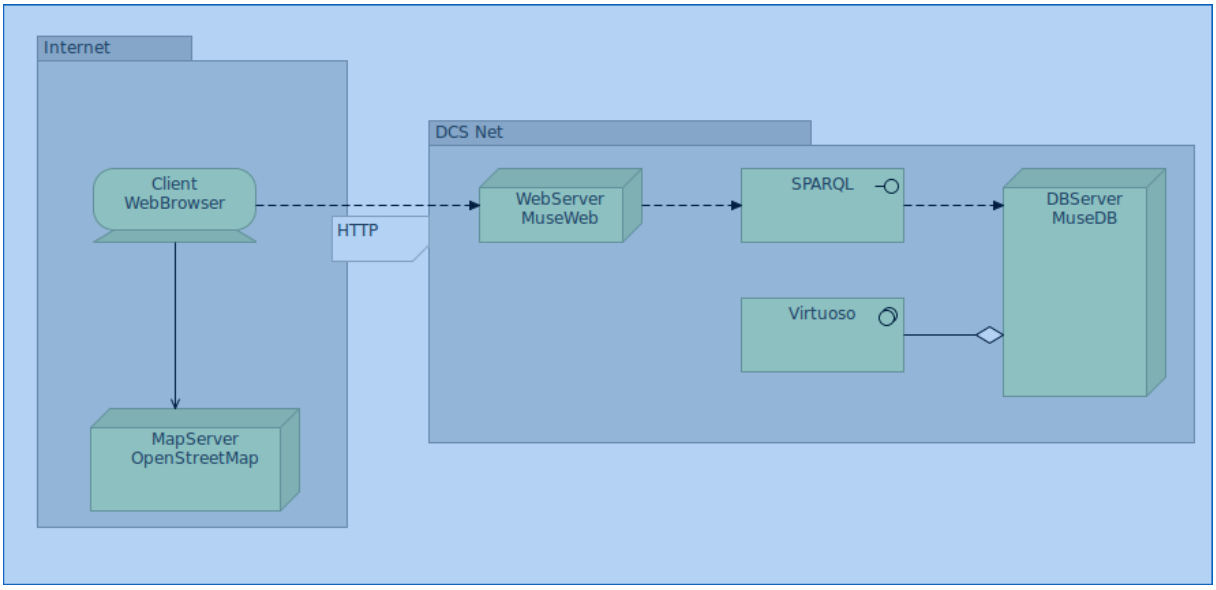

Figure 7: Physical architecture

```
.
├── app
│   ├── main
│   │   ├── boksplots
│   │   ├── datatypes
│   │   └── treelib
│   ├── static
│   │   ├── css
│   │   │   └── images
│   │   ├── custom-img
│   │   ├── img
│   │   └── js
│   └── templates
│       ├── 404.html
│       ├── 500.html
│       ├── base.html
│       ├── boksplot.html
│       ├── browseproperties.html
│       ├── index.html
│       ├── message.html
│       ├── nakedchloro.html
│       ├── nakedid.html
│       ├── nakedmap.html
│       ├── nakedresults.html
│       ├── plot.html
│       ├── search.html
│       ├── searchmessage.html
│       ├── searchResults.html
│       ├── showpage.html
│       ├── visualisation.html
│       └── visualisations.html
├── funktests
│   ├── test_basic_navigation-20180611T140516
│   │   └── index.html
│   ├── test_basic_navigation-20180613T094332
│   │   └── index.html
│   └── test_user_trial-20180613T101210
│       └── index.html
└── json
    └── boksgreeting.html
```

Figure 8: Front end structure