# Design Documentation

## 1. Design

```
### python3 -m bokeh serve --show ui_assWhole.py
import pyodbc
from functools import partial
from bokeh.io import output_file, show
from bokeh.layouts import row, column, layout, widgetbox
from bokeh.plotting import figure, curdoc
from bokeh.models import ColumnDataSource
from bokeh.models.widgets import Button, RadioButtonGroup, Select, Slider, TextInput
from bokeh.models.widgets import RadioGroup, CheckboxGroup, MultiSelect, Dropdown
from bokeh.models.widgets import DataTable, DateFormatter, TableColumn, Panel, Tabs
from bokeh.models.widgets import Paragraph
from bokeh.events import ButtonClick
from bokeh.palettes import Spectral6
from datetime import date
from random import randint
import string
from bokeh.core.properties import value
```

First are the import list. I combined sql.py and ui.py as a whole, then I imported pyodbc. I also imported value from bokeh.core.properties to get the input of “title_input” could be fetched using this method.

```
attr = dict(
    server = '10.20.213.10',
    database = 'csc1002',
    username = 'csc1002',
    password = 'csc1002',
    port = 1433,
    driver = 'ODBC Driver 13 for SQL Server'
)

conn_str = 'DRIVER={driver};'\
    'SERVER={server};'\
    'PORT={port};'\
    'DATABASE={database};'\
    'UID={username};'\
    'PWD={password}'

conn_str = conn_str.format(**attr)
```

Then, above are the information to connect to the server and fetch datas. By using ".format()" method, we fill the information into conn_str.

```
36  def obdcConnection():
37      try:
38          return pyodbc.connect(conn_str)
39      except Exception as e:
40          print(e)
41          quit()
42
43  cursor = obdcConnection().cursor()
```

Using try-except, we can define the function, odbcConnection( ), to acquire reliable database connection and print "e" when failed to connect. Using .cursor() method to define cursor, representing the whole database.

```
46  getDept = 'SELECT dept_name FROM lgu.student'
47  tsql = 'SELECT * FROM lgu.course'
48  gpa = ['A+', 'A', 'B+', 'B', 'C+', 'C', 'D+', 'D', 'F']
49  years = ["2015", "2016", "2017"]
50  colors = ["#c9d9d3", "#718dbf", "#e84d60"]
51
52  refresh = Button(label="Refresh")
53  btnGroupTitle = RadioButtonGroup(name='title', labels=["begins with...", "...contains...", "...ends with"], active=1)
54  btnGroupDept = RadioButtonGroup(name='dept', labels=["begins with...", "...contains...", "...ends with"], active=1)
55  paragraph = Paragraph(text="option")
56  optionGroup = RadioGroup(labels=["and", "or"], active=0, width=100, inline=True)
57  btnGroupLetters = RadioButtonGroup(labels=list(string.ascii_uppercase), active=-1)
58  title_input = TextInput(value="", title="Title:", placeholder="contains....")
59  dept_input = TextInput(value="", title="Department:", placeholder="contains....")
60
61  with cursor:    # get dept names
62      rows = cursor.execute(getDept).fetchall()
63      deptList = []
64      for row in rows:
65          if row.dept_name not in deptList:
66              deptList.append(row.dept_name)
67      deptList.sort()
68  select = Select(title="Department:", value=deptList[0], options=deptList)
69  deptname = widgetbox(select)
```

The strings, getDept and tsql, are the SQL instructions to get department names from lgu.student and lgu.course which would be executed in the SQL environment.

Line 48-50 are the basic information of the stack graph on the second tab.

Line 52-59 are given in ui.py and no modifications were performed.

Line 61-67 are aimed to get all department names by executing “getDept” in cursor then append all the names in the deptList.

Line 68-69 layout the selection list of all the departments on the second tab.

```
71  ###data table
72  columns = [
73      TableColumn(field = 'id', title = 'Course ID'),
74      TableColumn(field = 'title', title = 'Title'),
75      TableColumn(field = 'dept', title = 'Department'),
76      TableColumn(field = 'credit', title = 'Credit'),
77      TableColumn(field = 'instructor', title = 'Instructor')
78  ]
79  table_master = DataTable(source=ColumnDataSource(), columns = columns, width = 800, height = 200)
80  data = dict(
81      id = [],
82      title = [],
83      dept = [],
84      credit = [],
85      instructor = []
86  )
87  with cursor:
88      rows = cursor.execute(tsql).fetchall()
89      for row in rows:
90          data['id'].append('{id}'.format(id = row.course_id))
91          data['title'].append('{title}'.format(title = row.title))
92          data['dept'].append('{dept}'.format(dept = row.dept_name))
93          data['credit'].append('{credit}'.format(credit = row.credits))
94          data['instructor'].append('{instructor}'.format(instructor = row.instructor))
95
96  table_master.source.data = data
```

The lines above are set to create the default data table on the first tab.

Line 72-78 define the column of the data table; table_master in line 79 is the layout of the data table.

Line 80-94 define the dataset of the data table by fetching and appending the information.

Line 96 assigns the data source to the data table.

```
98  def select_handler(attr, old, new):
99      global data1, source
100     print('Select value changed from {} to {}'.format(old, new))
101     data1 = {'gpa' : gpa,
102         '2015'   : [],
103         '2016'   : [],
104         '2017'   : []}
105     with cursor:    # get related gpa
106         for i in ['2015','2016','2017']:
107             rows = cursor.execute('SELECT dept_name, gpa, year FROM lgu.student WHERE year = \'{}\' and dept_name = \'{}\''.format(i,
108             p0, p1, p2, p3, p4, p5, p6, p7, p8 = 0,0,0,0,0,0,0,0,0
109             for row in rows:
110                 if row.gpa == 'A+':
111                     p0 += 1
112                 elif row.gpa == 'A':
113                     p1 += 1
114                 elif row.gpa == 'B+':
115                     p2 += 1
116                 elif row.gpa == 'B':
117                     p3 += 1
118                 elif row.gpa == 'C+':
119                     p4 += 1
120                 elif row.gpa == 'C':
121                     p5 += 1
122                 elif row.gpa == 'D+':
123                     p6 += 1
124                 elif row.gpa == 'D':
125                     p7 += 1
126                 elif row.gpa == 'F':
127                     p8 += 1
128             for t in [p0, p1, p2, p3, p4, p5, p6, p7, p8]:
129                 data1[i].append(t)
130     source.data = data1
131
132 select.on_change('value',select_handler)
```

This part defined the select_handler function and used it within the .on_change( ) function to update the dataset of the gpa bar graph.

Line 100 aimed to give the feedback of such function has been run.

Line 101-104 empties the database, “data1”, to let the information not repeatedly appended.

```
107             rows = cursor.execute('SELECT dept_name, gpa, year FROM lgu.student WHERE year = \'{}\' and dept_name = \'{}\''.format(i, new)).fetchall()
```

Line 105-107 take a year from the list of the years and executed the string in line 107 (as above) and ran in the cursor to fetch department names, gpas, and years.

Line 108’s p0 - p8 represents the occurrences of the nine gpa standards: A+, A, B+, B, C+, C, D+, D, F. The initial value of them is 0.

Line 109-127 can add 1 to p0-p8 for each occurrence of the certain corresponding grade.

Line 128-130 append the gpa information into the dataset, “data”, and change the source of the bar graph.

By using .on_change( ) function in Line 132, the select_handler function can be called to update data by changing the selection list.

```
134    placeholder = {0:"begins with...", 1: "...contains...",2: "...ends with"}
136    def UpdatePlaceHolderT(idx):
137        print(idx, placeholder[idx])
138        title_input.placeholder = placeholder[idx]
139    def UpdatePlaceHolderD(idx):
140        print(idx, placeholder[idx])
141        dept_input.placeholder = placeholder[idx]
142
143    btnGroupTitle.on_click(UpdatePlaceHolderT)
144    btnGroupDept.on_click(UpdatePlaceHolderD)
```

The placeholder here is the placeholder of title_input and dept_input defined in line 58-59, representing the default display of the input box.

Line 136-141 are functions to update the placeholders.

Line 143-144 are aimed to trigger the functions, “UpdatePlaceHolderT” and “UpdatePlaceHolderD”, upon clicks of btnGroupTitle and btnGroupDept.

```
145  data1 = {'gpa' : gpa,
146      '2015'   : [],
147      '2016'   : [],
148      '2017'   : []}
149
150  with cursor:    # get related gpa
151      for i in ['2015','2016','2017']:
152          rows = cursor.execute('SELECT dept_name, gpa, year FROM lgu.student WHERE year = \'{}\' and dept_name = \'{}\''.format(i, deptList[0])).fetchall()
153          p0, p1, p2, p3, p4, p5, p6, p7, p8 = 0,0,0,0,0,0,0,0,0
154          for row in rows:
155              if row.gpa == 'A+':
156                  p0 += 1
157              elif row.gpa == 'A':
158                  p1 += 1
159              elif row.gpa == 'B+':
160                  p2 += 1
161              elif row.gpa == 'B':
162                  p3 += 1
163              elif row.gpa == 'C+':
164                  p4 += 1
165              elif row.gpa == 'C':
166                  p5 += 1
167              elif row.gpa == 'D+':
168                  p6 += 1
169              elif row.gpa == 'D':
170                  p7 += 1
171              elif row.gpa == 'F':
172                  p8 += 1
173          for t in [p0, p1, p2, p3, p4, p5, p6, p7, p8]:
174              data1[i].append(t)
175  source = ColumnDataSource(data=data1)
176  source.data = data1
```

Line 145-176 are basically the same as the select_handler function which is aimed to get the original default stack graph of gpa on the second tab.

The only two differences occurs in line 152 and line 175.

Line 152 let the default dept_name fetched from the SQL server as deptList[0], which is “Accounting”.

Line 175 get the default stack graph and make it ready to layout using ColumnDataSource( ) function.

```
178  def OnLetterSelected(idx):
179      letter = string.ascii_uppercase[idx]
180      data = dict(
181          id = [],
182          title = [],
183          dept = [],
184          credit = [],
185          instructor = []
186      )
187      with cursor:
188          tsql1 = "SELECT * FROM lgu.course WHERE title like '{}%'"
189          rows = cursor.execute(tsql1.format(string.ascii_uppercase[idx])).fetchall()
190          for row in rows:
191              data['id'].append('{id}'.format(id = row.course_id))
192              data['title'].append('{title}'.format(title = row.title))
193              data['dept'].append('{dept}'.format(dept = row.dept_name))
194              data['credit'].append('{credit}'.format(credit = row.credits))
195              data['instructor'].append('{instructor}'.format(instructor = row.instructor))
196      table_master.source.data = data
197      print(idx, letter)
198
199  btnGroupLetters.on_click(OnLetterSelected)
```

Line 178-199 defines the OnLetterSelected( ) function to let for each click on the top alphabet in the first tab, the courses on the data table can be changed into those titled with the corresponding letters.

Line 179 confirms the letter on which the user clicked.

Line 180-186 gets the empty list of all needed columns and combined those lists into a dictionary. And line 187-195 appends the information into the dictionary.

Line 196 changes the source of the dataset to the new dictionary, “data”.

Line 199 let the OnLetterSelected( ) function run after each click of the top alphabet in the first tab using .on_click( ) function.

```
201  def OnRefreshClick():
202      title = title_input.value
203      dept = dept_input.value
204      active = optionGroup.active
205      if active == 0:
206          active = 'and'
207      else:
208          active = 'or'
209      if title_input.placeholder == placeholder[0]:
210          title = title + '%'
211      elif title_input.placeholder == placeholder[2]:
212          title = '%' + title
213      else:
214          title = '%' + title + '%'
215      if dept_input.placeholder == placeholder[0]:
216          dept = dept + '%'
217      elif dept_input.placeholder == placeholder[2]:
218          dept = '%' + dept
219      else:
220          dept = '%' + dept + '%'
221      data = dict(
222          id = [],
223          title = [],
224          dept = [],
225          credit = [],
226          instructor = []
227      )
228      with cursor:
229          tsql1 = "SELECT * FROM lgu.course WHERE title like '{}' {} dept_name like '{}'"
230          rows = cursor.execute(tsql1.format(title,active,dept)).fetchall()
231          for row in rows:
232              data['id'].append('{id}'.format(id = row.course_id))
233              data['title'].append('{title}'.format(title = row.title))
234              data['dept'].append('{dept}'.format(dept = row.dept_name))
235              data['credit'].append('{credit}'.format(credit = row.credits))
236              data['instructor'].append('{instructor}'.format(instructor = row.instructor))
237      table_master.source.data = data
238      print('refresh title: {} Dept: {} Option:{}'.format(title,dept,active))
239
240  refresh.on_click(OnRefreshClick)
```

Line 201-240 defined the operations, specifically, searching according to the input of the user within the database of the cursor after the user press the “Refresh” button.

Line 202-204 fetch the information within the two input boxes and the selection between and & or.

Line 205-208 evaluate the information whether the conditions inputted by the user should be connected by “and” or “or” in SQL executions.

Line 209-220 interpret the user’s selection (from "begins with…", “… contains...", "...ends with”) into SQL language by adding “%” in different places.

Line 221-227 empty the dataset, “data”, to let the information not repeatedly appended.

Line 228-236 fetch and append the five informations row-by-row into the dataset by executing tsql1 in SQL environment.

Line 237 changes the source of the dataset to the new dictionary, “data”.

Line 240 is aimed to trigger the OnRefreshClick( ) function upon clicks of the “Refresh” button.

```
242  layout_query = layout(
243      [
244          [widgetbox(btnGroupLetters, width=1000)],
245          [widgetbox(btnGroupTitle), widgetbox(btnGroupDept)],
246          [widgetbox(title_input), widgetbox(paragraph, optionGroup, width=100), widgetbox(dept_input)],
247          [widgetbox(refresh, width=100)],
248          [widgetbox(table_master)]
249      ]
250  )
251
252  s1 = figure(x_range=gpa, plot_height=350, title="GPA situations by Year",
253          toolbar_location=None, tools="")
254
255  s1.vbar_stack(years, x='gpa', width=0.9, color=colors, source=source, legend=[value(x) for x in years], name=years)
256
257  layout_chart = layout (
258      [[widgetbox(select),s1]]
259  )
260
261  tab1 = Panel(child=layout_query, title="Course Info")
262  tab2 = Panel(child=layout_chart, title="Statistics")
263  tabs = Tabs(tabs=[tab1, tab2])
264
265  curdoc().add_root(tabs)
```

Line 242-265 are the core part of the program, which let the program’s layout and user interface display and functional.

On the first tab, according to “tab1” on line 261, the layouts are in the list “layout_query” defined between line 242-250 using layout( ) function.

Similarly, on the second tab, according to “tab2” on line 262, the layouts are in the list “layout_query” defined between line 257-259 using layout( ) function.

“s1” defined between line 252-255 is the stack graph of the overall GPA situation of the selected course title.

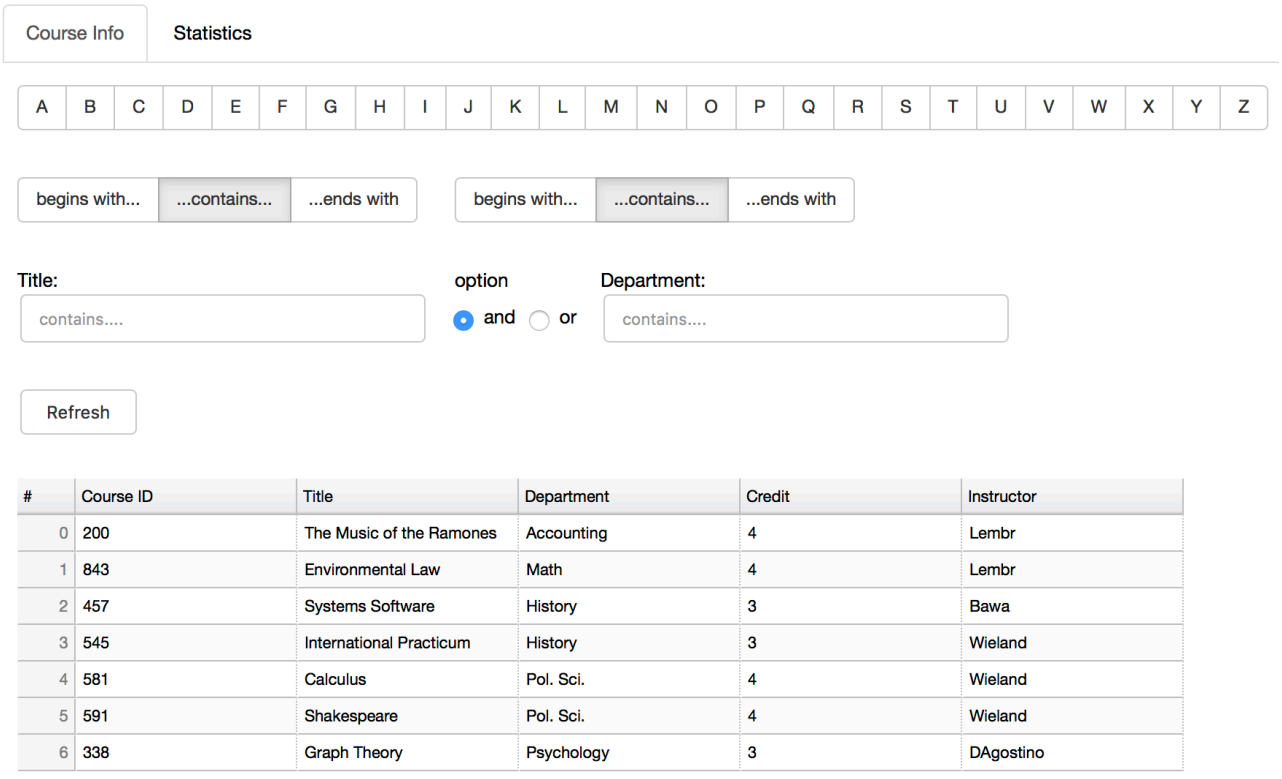
Line 265 is the last line of the program, letting all above display and functional, and the user can use this system eventually.

# 2. Test Plan

- This test plan is focused on the following six perspectives:
    1. The default layouts of the dataset;
    2. The update of the placeholders with the change of button groups of search filter methods ("begins with…", "…contains...", "...ends with");
    3. The update of the data table with the selection change of alphabetical button group;
    4. The update of the data table with each press of the "Refresh" button;
    5. The default layout of the stacked bar chart;
    6. The update of the stacked bar chart corresponding to each change of selection in the selection list.

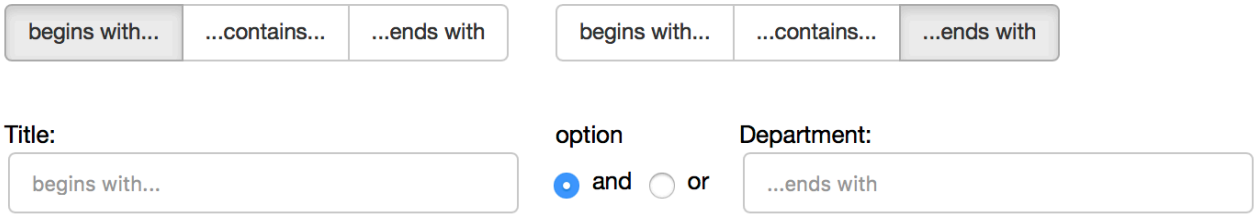
## 1. ***Default layouts of the dataset***

The default layout of the dataset is shown as follows:

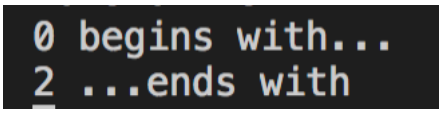



| # | Course ID | Title | Department | Credit | Instructor |
|---|---|---|---|---|---|
| 0 | 200 | The Music of the Ramones | Accounting | 4 | Lembr |
| 1 | 843 | Environmental Law | Math | 4 | Lembr |
| 2 | 457 | Systems Software | History | 3 | Bawa |
| 3 | 545 | International Practicum | History | 3 | Wieland |
| 4 | 581 | Calculus | Pol. Sci. | 4 | Wieland |
| 5 | 591 | Shakespeare | Pol. Sci. | 4 | Wieland |
| 6 | 338 | Graph Theory | Psychology | 3 | DAgostino |

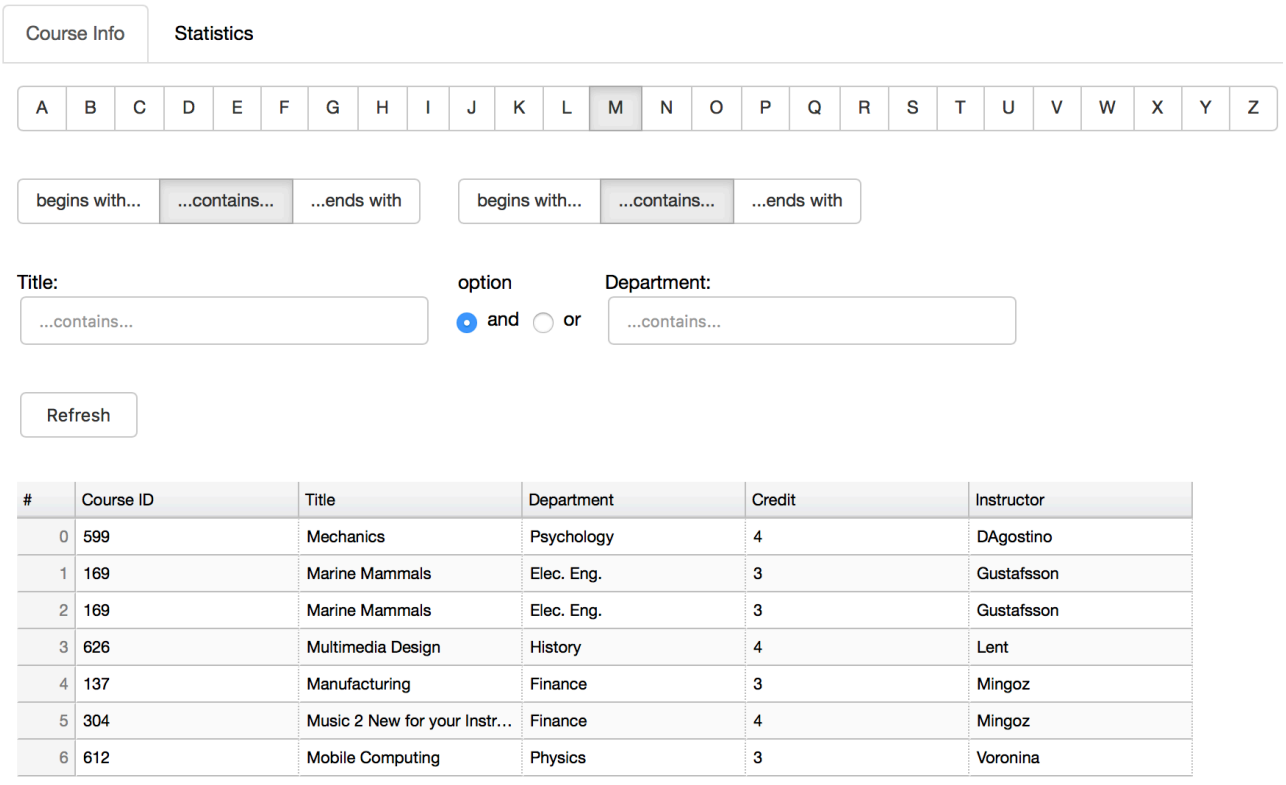
## 2. *Update of the placeholders*




The placeholder can change with clicks of button groups of search filter methods within this test, and the layout inside the terminal is as shown below:




## 3. *Update of the data table ( I )*

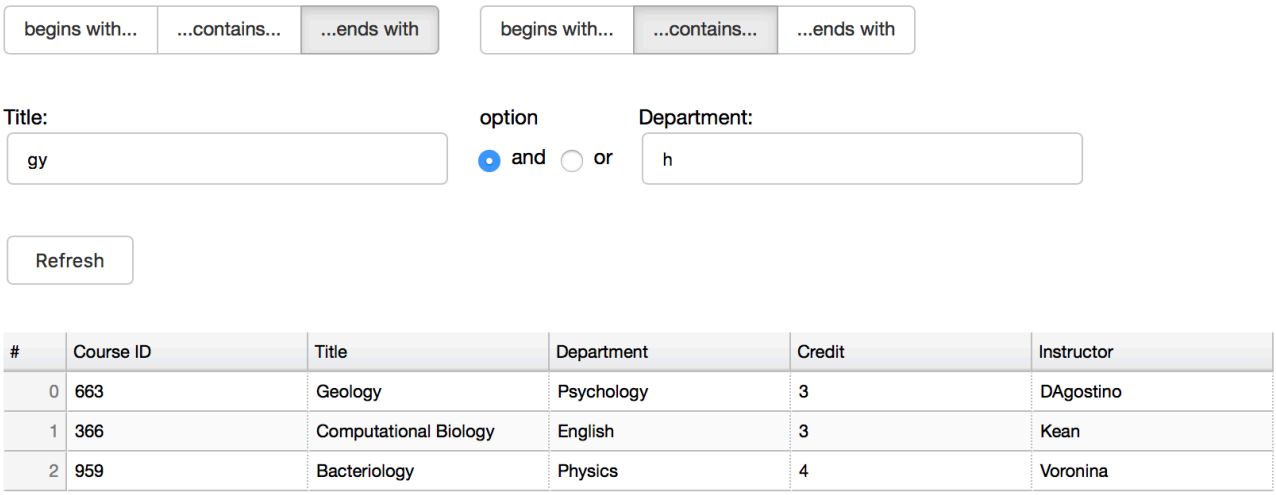



| # | Course ID | Title | Department | Credit | Instructor |
|---|---|---|---|---|---|
| 0 | 599 | Mechanics | Psychology | 4 | DAgostino |
| 1 | 169 | Marine Mammals | Elec. Eng. | 3 | Gustafsson |
| 2 | 169 | Marine Mammals | Elec. Eng. | 3 | Gustafsson |
| 3 | 626 | Multimedia Design | History | 4 | Lent |
| 4 | 137 | Manufacturing | Finance | 3 | Mingoz |
| 5 | 304 | Music 2 New for your Instr... | Finance | 4 | Mingoz |
| 6 | 612 | Mobile Computing | Physics | 3 | Voronina |

After clicking “M” in the alphabetical button group, the data table updates to let all the course titles begin with letter “M”. The function is well operated within this test. And the layout inside the terminal is as shown below:

## 4. ***Update of the data table ( II )***

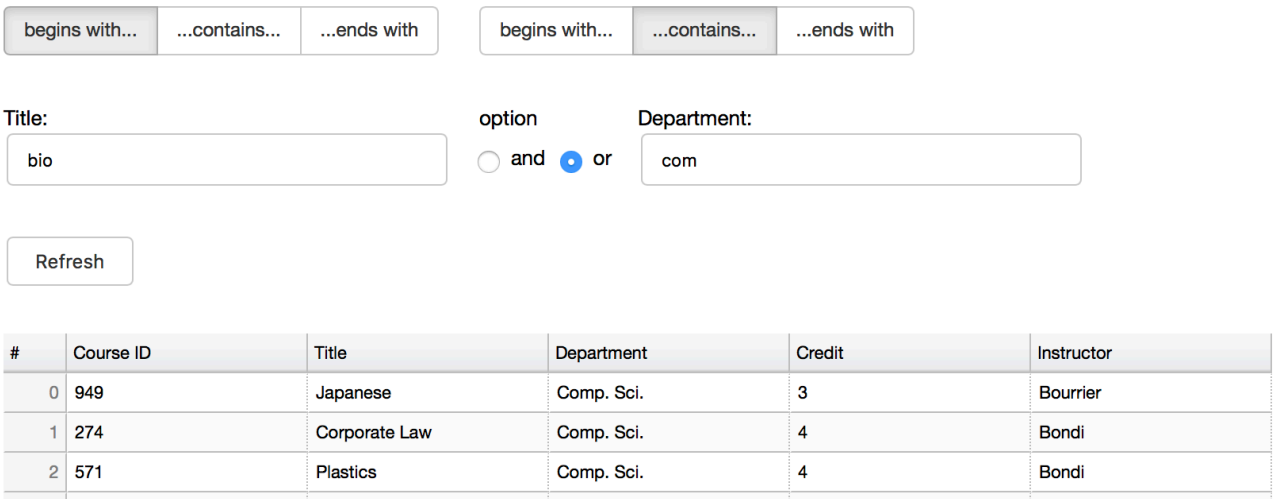
- *The first set of test:*




| # | Course ID | Title | Department | Credit | Instructor |
|---|---|---|---|---|---|
| 0 | 663 | Geology | Psychology | 3 | DAgostino |
| 1 | 366 | Computational Biology | English | 3 | Kean |
| 2 | 959 | Bacteriology | Physics | 4 | Voronina |

Corresponding layout inside the terminal:

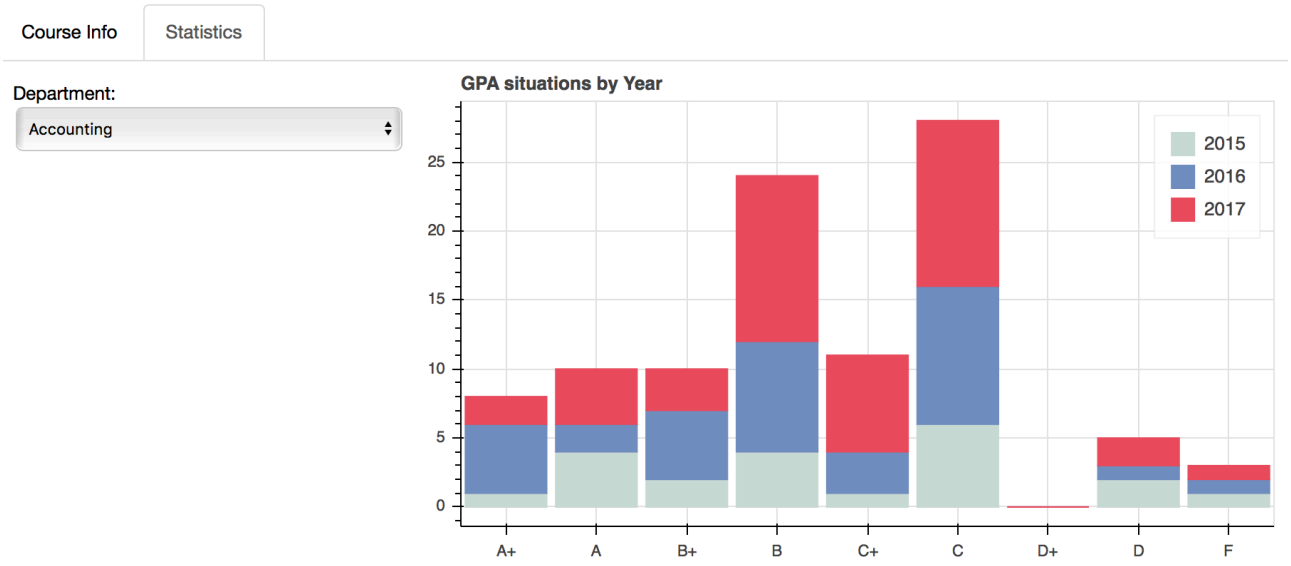

- *The second set of test:*




| # | Course ID | Title | Department | Credit | Instructor |
|---|---|---|---|---|---|
| 0 | 949 | Japanese | Comp. Sci. | 3 | Bourrier |
| 1 | 274 | Corporate Law | Comp. Sci. | 4 | Bondi |
| 2 | 571 | Plastics | Comp. Sci. | 4 | Bondi |
| 3 | 747 | International Practicum | Comp. Sci. | 4 | Bondi |
| 4 | 445 | Biostatistics | Finance | 3 | Mingoz |
| 5 | 875 | Bioinformatics | Cybernetics | 3 | Pimenta |

Corresponding layout inside the terminal:

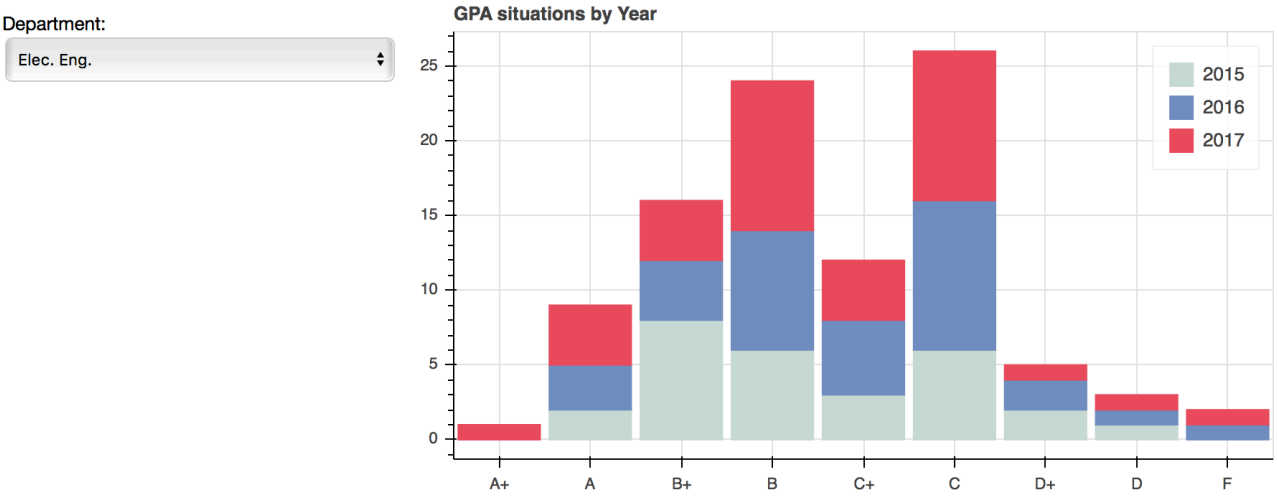
## 5. ***Default layout of the stacked bar chart***




The default layout of the stacked bar chart is the GPA situation of Accounting (on the first place of the list of departments). This function is well operated within this test.

## 6. ***Update of the stacked bar chart***

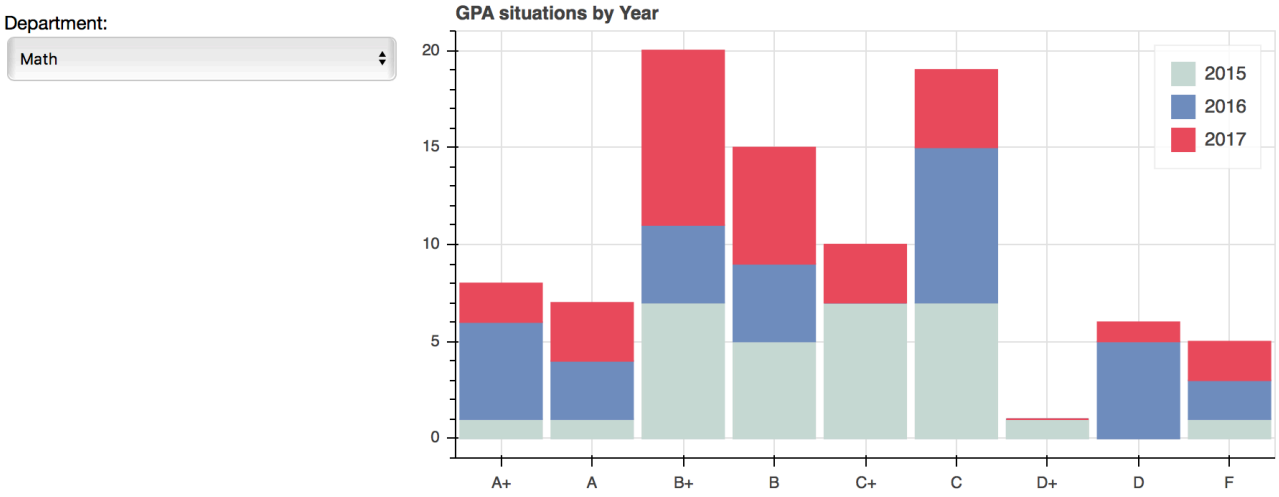
- *The first set of test:*




Corresponding layout inside the terminal:

```
Select value changed from Accounting to Elec. Eng.
```

- *The second set of test:*




Corresponding layout inside the terminal:

```
Select value changed from Elec. Eng. to Math
```

Therefore, the update of the stacked bar chart corresponding to each change of selection in the selection list is well functional within this test.