



PyF A dataflow processing framework

Dataflow processing, mining, transforming and reporting using Python.

Presented by :
Jonathan Schemoul



What is dataflow programming ?

- In the 70s: John Paul Morrison in Canada
 - Bank applications
- We process data one by one
 - Items pass from one block to another, creating a flow
- We transform (adapt) items in the flow like in a production chain.



What is dataflow programming ?

In the theory :

- All the blocks (components) make a network
- Components have input and output
- Network is managed by a scheduler

Checks network state, manage processes

- Data sent as information packets

Dataflow programming with Python

We have generators

Code execution gets resumed at next item

What if we tried to use that for dataflow programming ?

```
>>> def my_generator(size=100) :  
...     for i in range(size) :  
...         print « I continue », i  
...         yield dict(iteration=i,  
...                     answer=42)  
...  
>>> iterator = my_generator(size=5)  
>>> print repr(iterator.next())  
I continue 0  
{'iteration': 0, 'answer': 42}  
>>> for value in iterator :  
...     print repr(value)  
I continue 1  
{'iteration': 1, 'answer': 42}  
I continue 2  
{'iteration': 2, 'answer': 42}  
I continue 3  
{'iteration': 3, 'answer': 42}  
I continue 4  
{'iteration': 4, 'answer': 42}
```

Chaining generators

To use generators for dataflow programming...

We define the blocks, with generators

- We pass the iterator to another generator, making a pipe
- We « adapt » the flow with each block

```
>>> def my_adapter(flux) :
...     for v in flux:
...         print « I received », v
...         yield dict(value=v,
...                     double=v*2)
...
>>> it1 = xrange(5) # 0 to 5
>>> it2 = my_adapter(it1)
>>> def my_consumer(flux_adapted) :
...     for nv in flux_adapted:
...         print nv['double']
...         yield True # It's ok.
...
>>> it3 = my_consumer(it2)
>>> for value in it3:
...     if value is True:
...         print 'ok so far'
...     else:
...         print 'not ok'
I received 0
0
ok so far
I received 1
2
ok so far
[...]
```



Just a few words

- Blocks (components) in a process make a network
- Each component has input and output ports
- The network is managed using generators (lazy)
- Transferred data can be either standard python objects or « Information Packets ».



Benefits using lazy flow programming with PyF

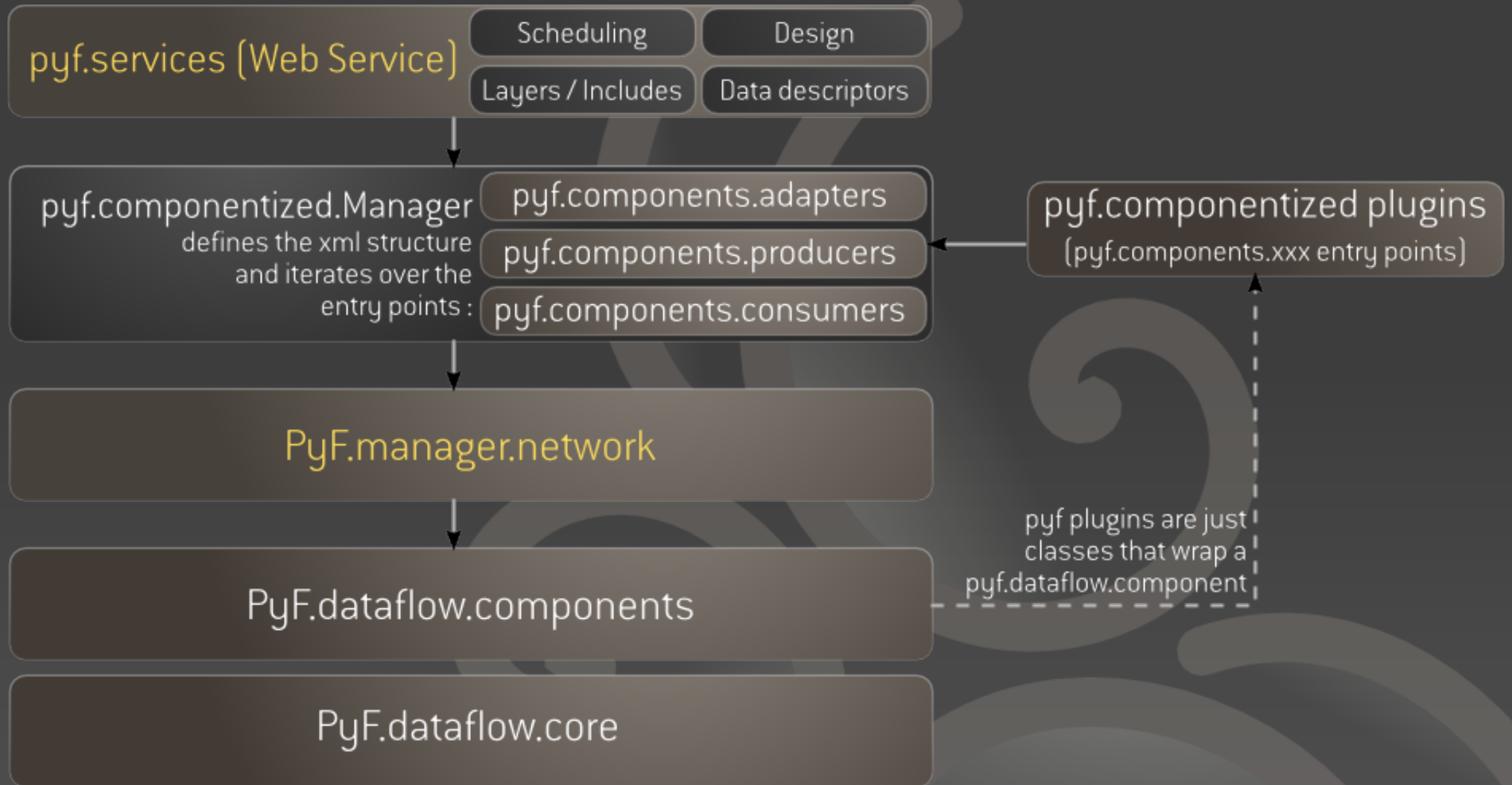
- RAM usage is kept low even with millions of items
- Easy to learn programming paradigm (you take one item, you give one item or several, or none, you choose! :-))
- All python objects can be used (not only packets)
- No threads
- Just standard plain python (no stackless)
- Need to scale ? Just ask for multiprocessing !



PyF in a few words

- Development framework and environment for dataflow programming
- Based on Zflow (the low-level core is a « friendly fork »)
- Much more than the basis : « batteries included ».

PyF Architecture

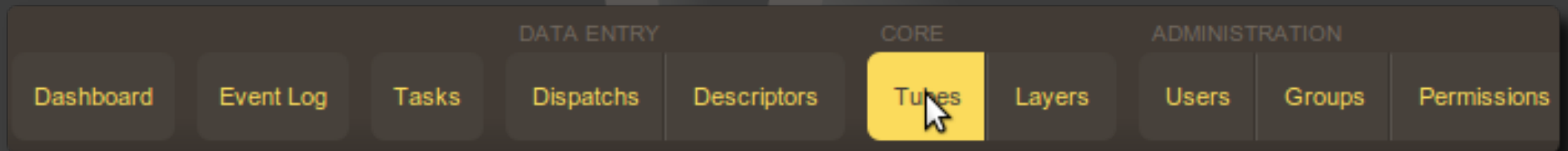




PyF in detail

- Plugins system
 - A lot of existing plugins
 - Extraction (website, rss, sql alchemy...),
 - Adaptation (summarizing, attribute getters, ...),
 - Writing (csv, xml, fixed-length, xlsx, pdf with xhtml, odt or rml templating, etc.)
 - Python (and soon javascript and ruby) code can be inserted directly in tube source
- A lot of other tools (visual designer, scheduling, event logs...)

The Web Service



Creating a tube

Info

Name:

Display Name:

Active

Needs Source

Components

- producer plugin
- consumer plugin
- producer code
- consumer code
- adapter code
- DescriptorSource
- WebExtractor
- SetAttributes
- DefaultEncoder

Controls

producer code

unique name

Name

Config

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10
- 11
- 12
- 13
- 14
- 15
- 16
- 17
- 18
- 19
- 20

Using Code

producer code

▶ Joiner Info

Name

Config

```
1 class User(object):
2     def __init__(self, name, email, level):
3         self.name = name
4         self.email = email
5         self.level = level
6
7     def get_source():
8         for index in range(0, 10):
9             yield User(
10                "John%02d" % index,
11                "john%02d" % index,
12                ['high', 'low'][index % 2])
13
14
15
16
17
18
19
20
21
22
23
```

adapter code

▶ Joiner Info

▶ Advanced

Name

Config [Add](#)

```
1 from itertools import groupby
2 from operator import attrgetter
3 from pyf.transport import Packet
4
5 def aggregate_lines(lines):
6     for account_code, acc_lines in groupby(lines, key=attrgetter('account_code')):
7         yield Packet(dict(account_code=account_code,
8                            lines=list(acc_lines)))
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
```

Or plugins

pyf.components.adapters.simple_filter

▶ **Joiner Info**

▶ **Advanced**

Name

Filter expression

pyf.components.adapters.summarizer

▶ **Joiner Info**

▶ **Advanced**

Name

Attribute	<input type="text" value="total_amount"/>
Summary Type	<input type="text" value="sum"/>
Data Getter	<input type="text" value="attribute"/>
Getter	<input type="text" value="amount"/>
	remove

Attribute	<input type="text" value="average_amount"/>
Summary Type	<input type="text" value="average"/>
Data Getter	<input type="text" value="attribute"/>
Getter	<input type="text" value="amount"/>
	remove

[Add](#)

Yield Items (or just the summary at the end)

Attribute of summary

Link and arrange your nodes...

21
22
23



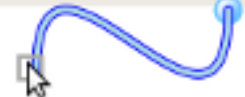
pyf.components.adapters.simple_filter X

► Joiner Info

Name

Filter expression

21
22
23



pyf.components.adapters.simple_filter X

► Joiner Info

Name

Filter expression

21
22
23



pyf.components.adapters.simple_filter X

► Joiner Info

Name

Filter expression

▼ **Advanced**

Separate Process

```
producer code
> Joiner Info
Name: source1
Config: Add
1 class User(object):
2     def __init__(self, name, email, level):
3         self.name = name
4         self.email = email
5         self.level = level
6     def get_source():
7         for index in range(0, 10):
8             yield User(
9                 "John04d" % index,
10                "john04d@some-where.com" % index,
11                ["high", "low", "high", "high", "low"][index%5])
12
13
14
15
16
17
18
19
20
21
22
23
```

pyf.components.adapters.simple_filter

> Joiner Info

Name: filter1

Filter expression: item.level == "high"

```
adapter code
> Joiner Info
Name: filter2
Config: Add
1 def low_or_med_items():
2     for item in items:
3         if item.level == "low" or item.level == "med":
4             yield item
5         else:
6             yield Ellipsis
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
```

pyf.components.adapters.compute_attrbu

> Joiner Info

Name: adapter1

Attribute: numeric_level

Type: eval

Value: 10

Add

```
adapter code
> Joiner Info
Name: adapter2
Config: Add
1 def set_numeric_level(items):
2     for item in items:
3         item.numeric_level = 0
4         yield item
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
```

pyf.components.consumers.csvwriter

> Joiner Info

Name: csvoutput

Encoding: UTF-8

Target filename: user_levels.csv

Delimiter: ;

Write header line:

Title: name

Attribute: Source object attribute

Renderer: Renderer eval (optional)

Title: email

Attribute: Source object attribute

Renderer: Renderer eval (optional)

Title: level

Attribute: numeric_level

Renderer: Renderer eval (optional)

Add


```
producer code
> Joiner Info
> Advanced
Name:
Count: 1
1 from pyf.transport import Packet
2 from random import randint
3 from itertools import cycle
4 count = 1000
5
6 pnames = ['David', 'Robert', 'Edward',
7           'Mary', 'Susan', 'William', 'Henry',
8           'James', 'Jonathan', 'Lisa', 'Julia',
9           'Charles', 'Margaret', 'James', 'William',
10          'Thomas', 'Susan', 'Robert', 'Edward']
11 names = []
12
13 combinations = [(pnames, name) for pnames, name in names]
14 combinations = combinations
15
16 clients = cycle(combinations)
17
18 def generate_customers():
19     for name in names:
20         for pnames, name in clients:
21             packet = Packet(name)
22             yield Packet(name, count=randint(1,100))
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
```

pyf.components.adapters.compute_adapter

Attribute	CF_name	Value	cf_name
Attribute	Type	Value	cf_name
Attribute	Value	Value	cf_name
Attribute	Value	Value	cf_name
Attribute	Value	Value	cf_name
Attribute	Value	Value	cf_name

```
adapter code
> Joiner Info
> Advanced
Name: lines_aggregate
Count: 1
1 from itertools import groupby
2 from operator import attrgetter
3 from pyf.transport import Packet
4
5 def aggregate_lines(lines):
6     for key, group in groupby(lines, key=attrgetter('customer_id')):
7         yield Packet(customer_id=key, lines=list(group))
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
```

pyf.components.consumers.coverletter

Attribute	lines_aggregate	Value	lines_aggregate
Attribute	Type	Value	lines_aggregate
Attribute	Value	Value	lines_aggregate
Attribute	Value	Value	lines_aggregate
Attribute	Value	Value	lines_aggregate
Attribute	Value	Value	lines_aggregate

pyf.components.adapters.summarizer

Attribute	lines_aggregate	Value	lines_aggregate
Attribute	Type	Value	lines_aggregate
Attribute	Value	Value	lines_aggregate
Attribute	Value	Value	lines_aggregate
Attribute	Value	Value	lines_aggregate
Attribute	Value	Value	lines_aggregate

pyf.components.consumers.coverletter

Attribute	lines_aggregate	Value	lines_aggregate
Attribute	Type	Value	lines_aggregate
Attribute	Value	Value	lines_aggregate
Attribute	Value	Value	lines_aggregate
Attribute	Value	Value	lines_aggregate
Attribute	Value	Value	lines_aggregate

```
pyf.components.adapters.simple_line
> Joiner Info
> Advanced
Name: lines_aggregate
Count: 1
1 from itertools import groupby
2 from operator import attrgetter
3 from pyf.transport import Packet
4
5 def aggregate_lines(lines):
6     for key, group in groupby(lines, key=attrgetter('customer_id')):
7         yield Packet(customer_id=key, lines=list(group))
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
```

```
adapter code
> Joiner Info
> Advanced
Name:
Count: 1
1 def adapt(records):
2     for record in records:
3         customer = record['customer_id']
4         customer_lines = record['lines_aggregate']['lines']
5         yield customer
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
```

pyf.components.adapters.compute_adapter

Attribute	CF_name	Value	cf_name
Attribute	Type	Value	cf_name
Attribute	Value	Value	cf_name
Attribute	Value	Value	cf_name
Attribute	Value	Value	cf_name
Attribute	Value	Value	cf_name

pyf.components.consumers.coverletter

Attribute	customer_id	Value	customer_id
Attribute	Type	Value	customer_id
Attribute	Value	Value	customer_id
Attribute	Value	Value	customer_id
Attribute	Value	Value	customer_id
Attribute	Value	Value	customer_id

pyf.components.adapters.simple_line

Attribute	lines_aggregate	Value	lines_aggregate
Attribute	Type	Value	lines_aggregate
Attribute	Value	Value	lines_aggregate
Attribute	Value	Value	lines_aggregate
Attribute	Value	Value	lines_aggregate
Attribute	Value	Value	lines_aggregate

```
pyf.components.consumers.adapter
> Joiner Info
> Advanced
Name:
Count: 1
1 from pyf.transport import Packet
2
3 def adapt(records):
4     for record in records:
5         customer_id = record['customer_id']
6         customer_lines = record['lines_aggregate']['lines']
7         yield Packet(customer_id=customer_id, lines=customer_lines)
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
```

- Thank you !
- Training is tomorrow at 14:30 in Pizza Napoli
- For more information :
 - <http://www.pyfproject.org/>*
 - [#pyf](#) on irc.freenode.net*
 - <http://groups.google.com/group/pyf-users>*
- *Business Cards available*



PyF Sponsors



Consulting
& Innovation
Group

<http://www.xcg-consulting.fr>



Service et conseil pour les systèmes d'information

<http://www.jmsinfor.com>



Training Info

- Tomorrow at 14:30 (2:30 PM) at Pizza Napoli
- Please have :
 - Under linux : libxml2-dev libxslt-dev
 - Installed virtualenv (« easy_install -UZ virtualenv »)
- Recommended :
 - Predownload pyf in a new virtualenv:
 - « easy_install -UZ pyf[fullstack] »