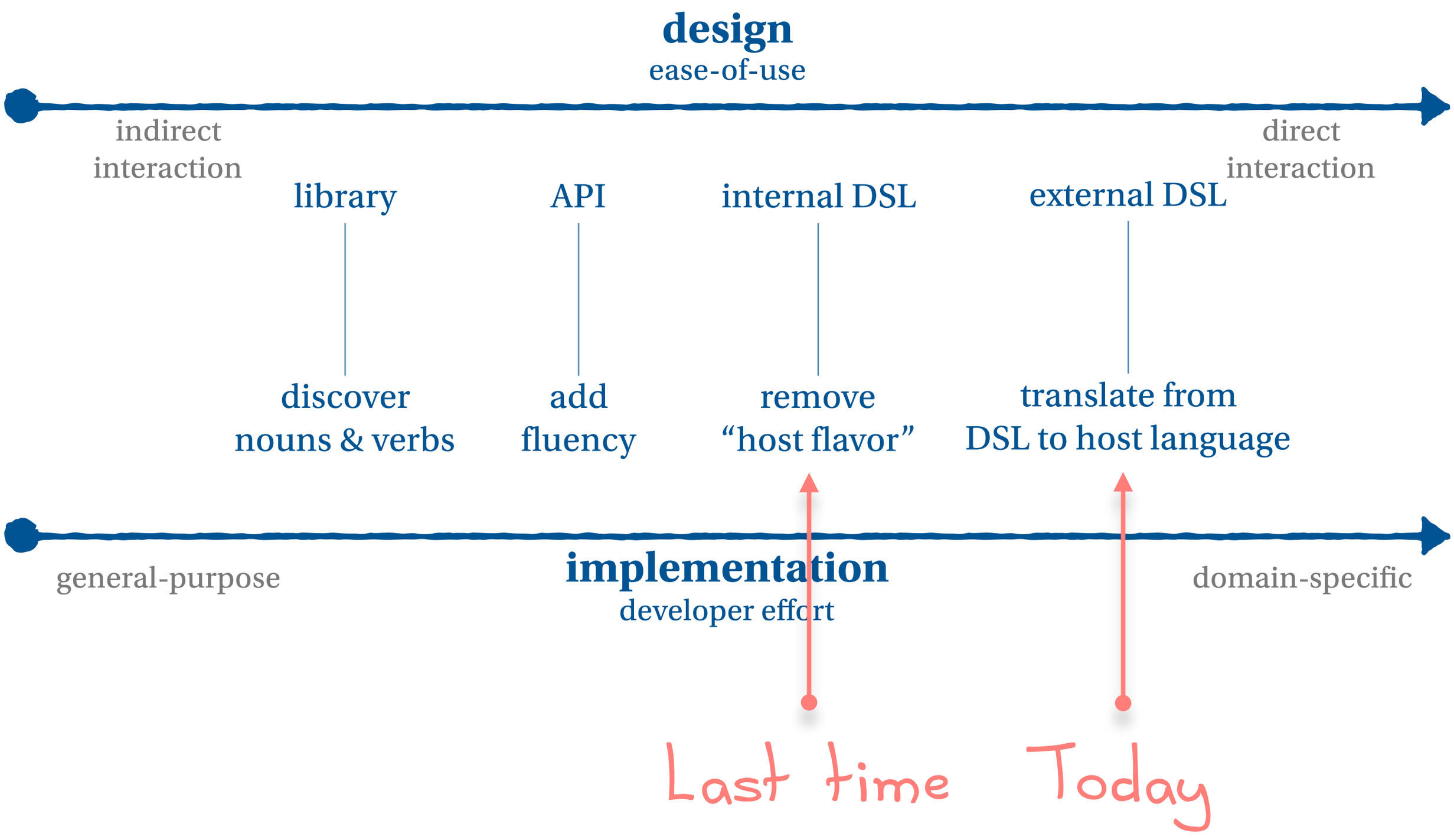
Today: implementation strategies

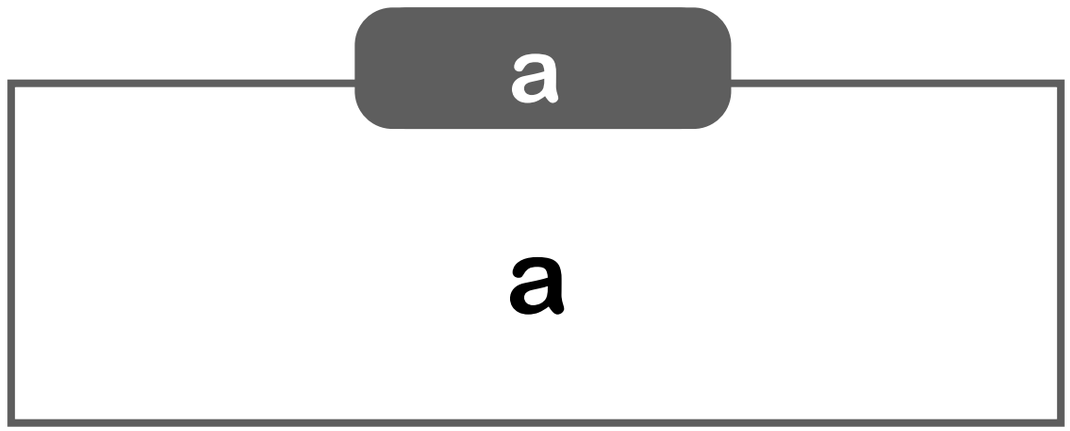


Regular expressions

for string-matching

Literal matching

one character matches

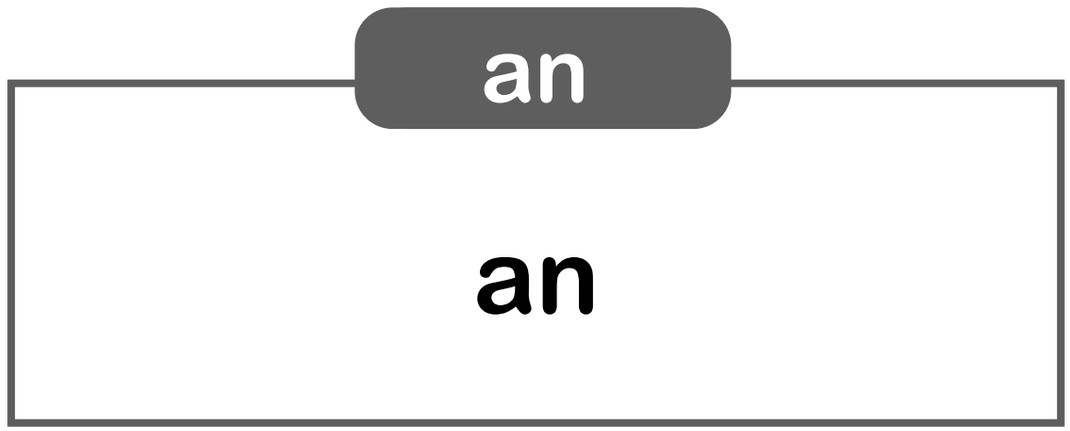



```
object Program extends App {
  val pattern = """a""".r
  val string = "banana"
  pattern.findAllIn(string) foreach println
}
```

```
a
a
a
```

Sequential matching

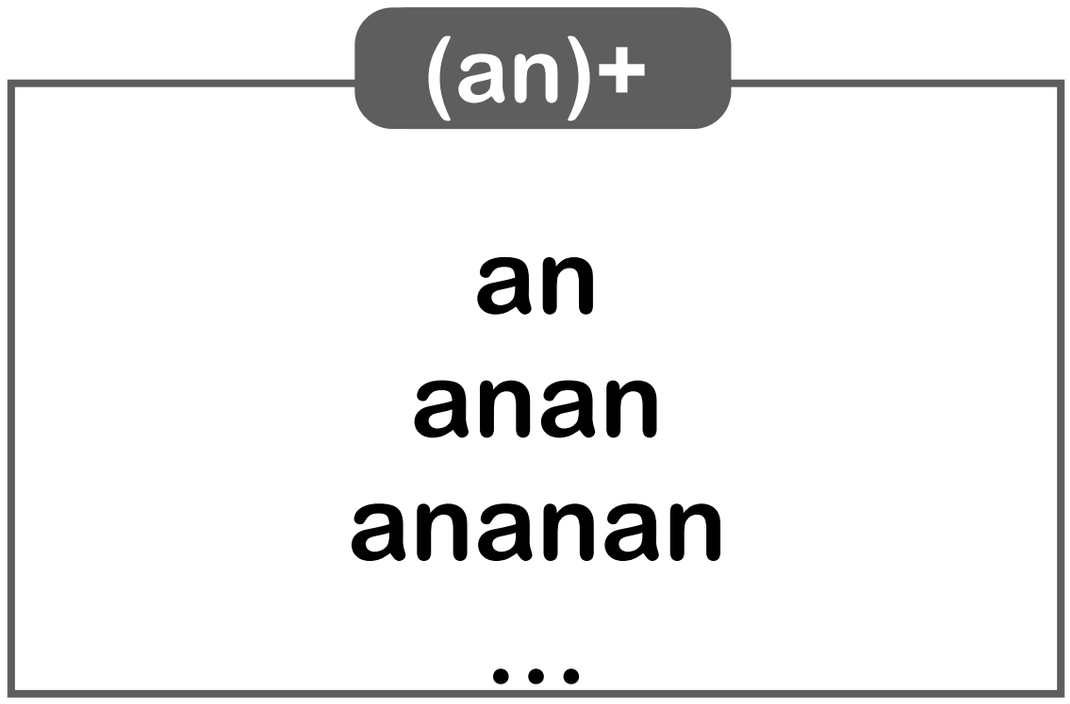
a sequence of characters matches




```
object Program extends App {
  val pattern = """an""".r
  val string = "banana"
  pattern.findAllIn(string) foreach println
}
```

```
an
an
```

Repetition: one or more



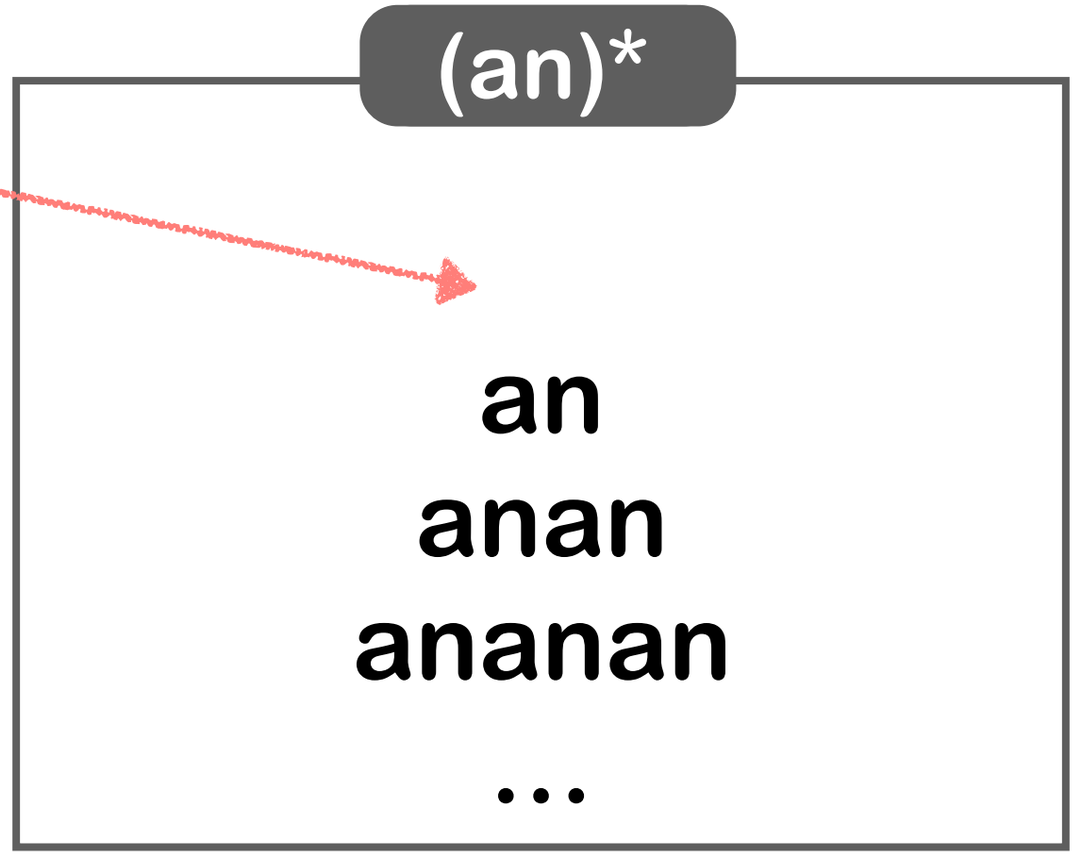

```
object Program extends App {
  val pattern = """(an)+""".r
  val string = "banana"
  pattern.findAllIn(string) foreach println
}
```

```
anan
```

finds the longest match that doesn't overlap with any other match

Repetition: zero or more

empty string
is a match




```
object Program extends App {
  val pattern = """(an)*""".r
  val string = "banana"
  pattern.findAllIn(string) foreach println
}
```

```
anan
```

Repetition: zero or one

a(b)?a

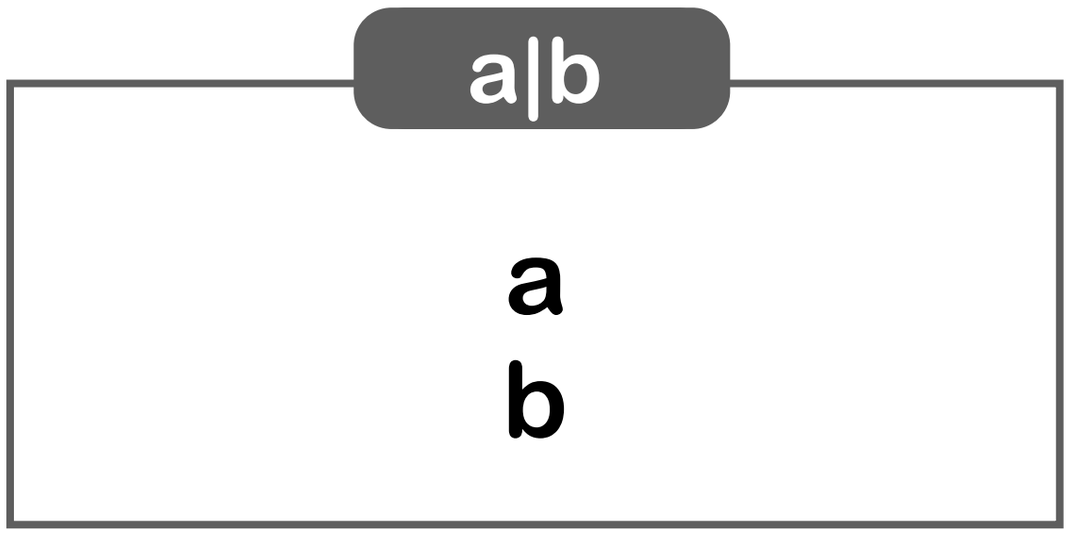
aa
aba

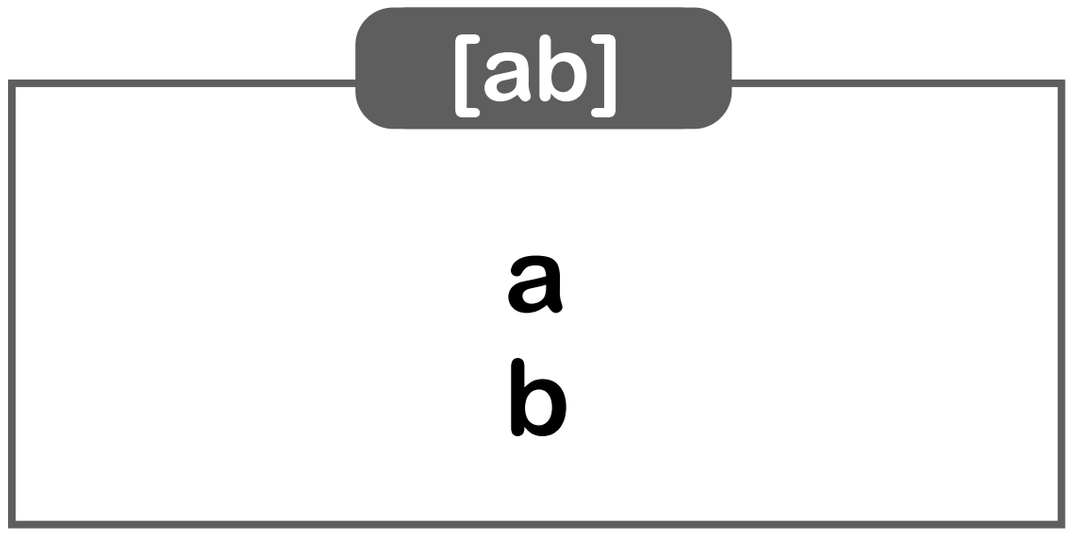
```
object Program extends App {
  val pattern = """a(b)?a""".r
  val string = "aa"
  pattern.findAllIn(string) foreach println
}
```

```
aa
```

Alternatives

two ways to say: “this or that”






```
object Program extends App {
  val pattern = """a|b""".r
  val string = "banana"
  pattern.findAllIn(string) foreach println
}
```

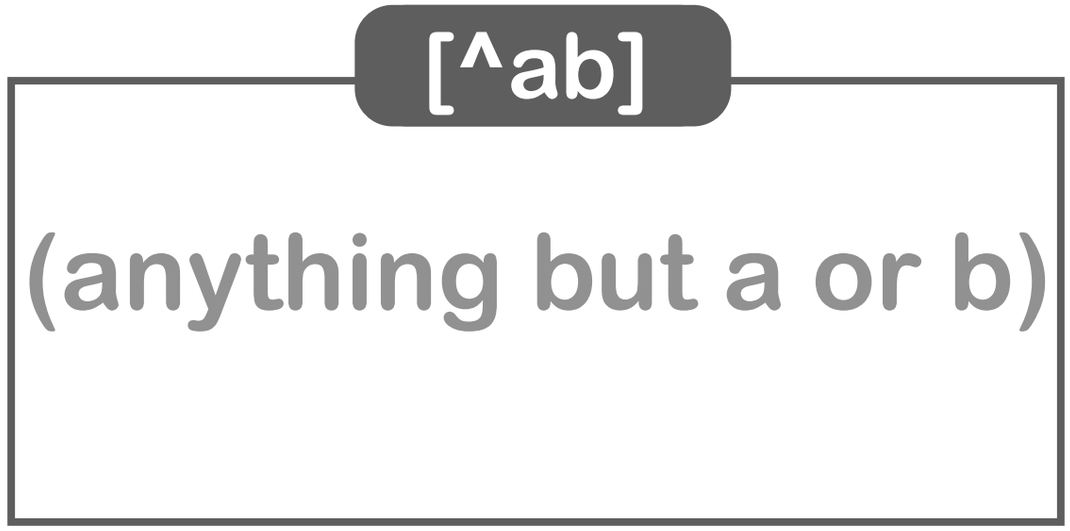
```
b
a
a
a
```

```
object Program extends App {
  val pattern = """[ab]""".r
  val string = "banana"
  pattern.findAllIn(string) foreach println
}
```

```
b
a
a
a
```

Negation

not these characters




```
object Program extends App {
  val pattern = """[^ab]""".r
  val string = "banana"
  pattern.findAllIn(string) foreach println
}
```

```
n
n
```

Character classes

multiple ways to say: any character in this range

[0-9]

0
1
2
3
4
5
6
7
8
9

```
object Program extends App {
  val pattern = """[0-9]""".r
  val string = "abc123forty7"
  pattern.findAllIn(string) foreach println
}
```

```
1
2
3
7
```

\d

0
1
2
3
4
5
6
7
8
9

```
object Program extends App {
  val pattern = """\d""".r
  val string = "abc123forty7"
  pattern.findAllIn(string) foreach println
}
```

```
1
2
3
7
```

More character classes

Decimals	**\d**	**[0-9]**
	\D	**[^0-9]**
Whitespace	**\s**	**[\t\n\r\f\v]**
	\S	**[^ \t\n\r\f\v]**
Alpha-Numeric	**\w**	**[a-zA-Z0-9_]**
	\W	**[^a-zA-Z0-9_]**
Backslash	****	**[\\]**
Any Character	**.**	**(everything)**

Matching an entire string

```
object Program extends App {
  val pattern = """^\w+\d+$""".r
  val string = "abc123"
  pattern.findAllIn(string) foreach println
}
```

```
abc123
```

```
object Program extends App {
  val pattern = """^\w+\d+$""".r
  val string = "abc123a"
  pattern.findAllIn(string) foreach println
}
```

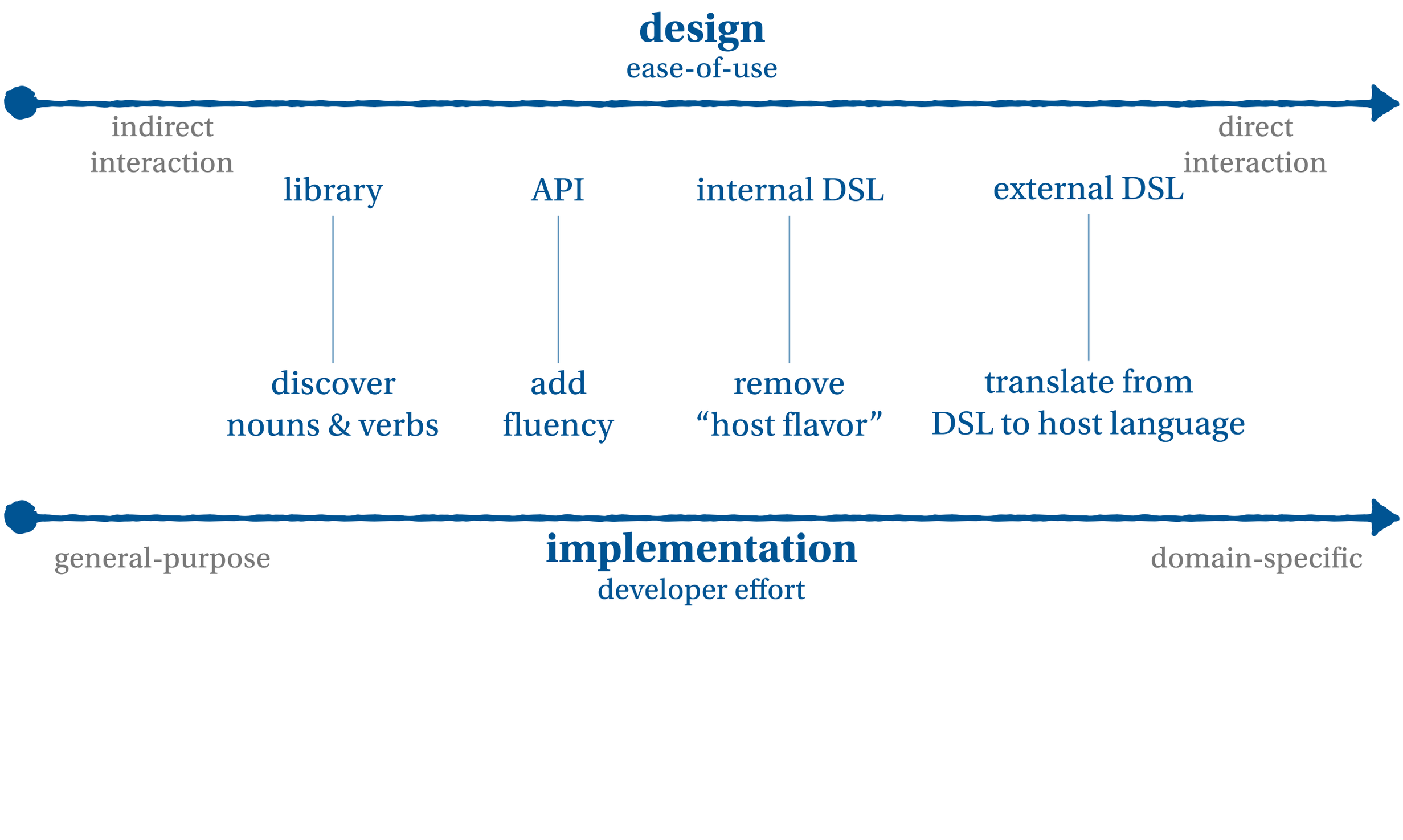
Groups and matching

Scala has some special magic that helps us pull matches out of strings

```
object Program extends App {
  val pattern = """^([a-z]+)(\d+)$""".r
  val string = "abc123"
  val pattern(letters, numbers) = string
  println(s"Letters: $letters")
  println(s"Numbers: $numbers")
}
```

```
Letters: abc
Numbers: 123
```

DSL implementation strategies



Is it a DSL?

Programming Language

Describe something to a computer.

General-purpose

Allow a professional programmer to write an arbitrary program.

restrict focus

Domain-specific

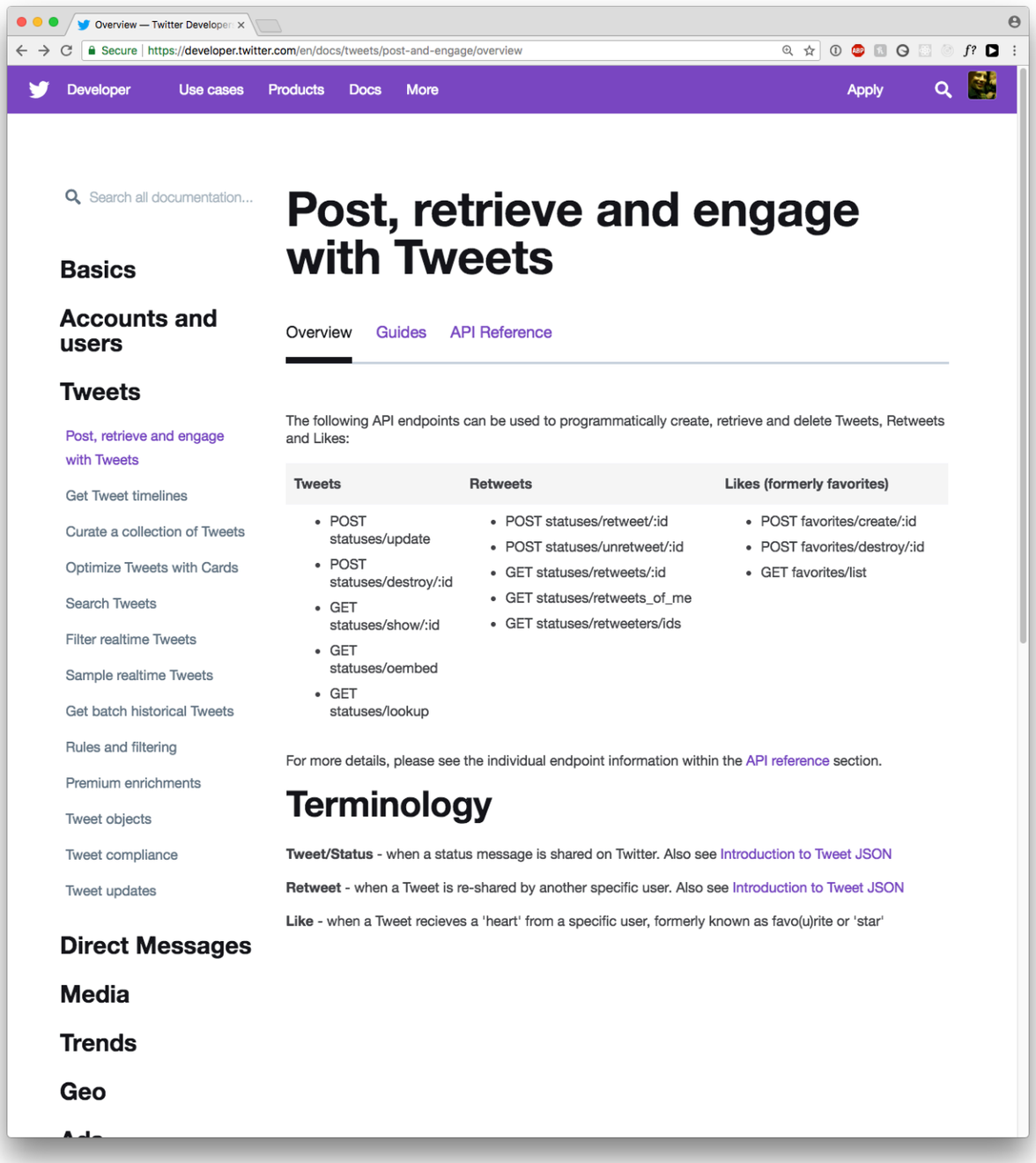
Allow a **domain expert** to interact directly with their domain.

We should have good answers for all these questions

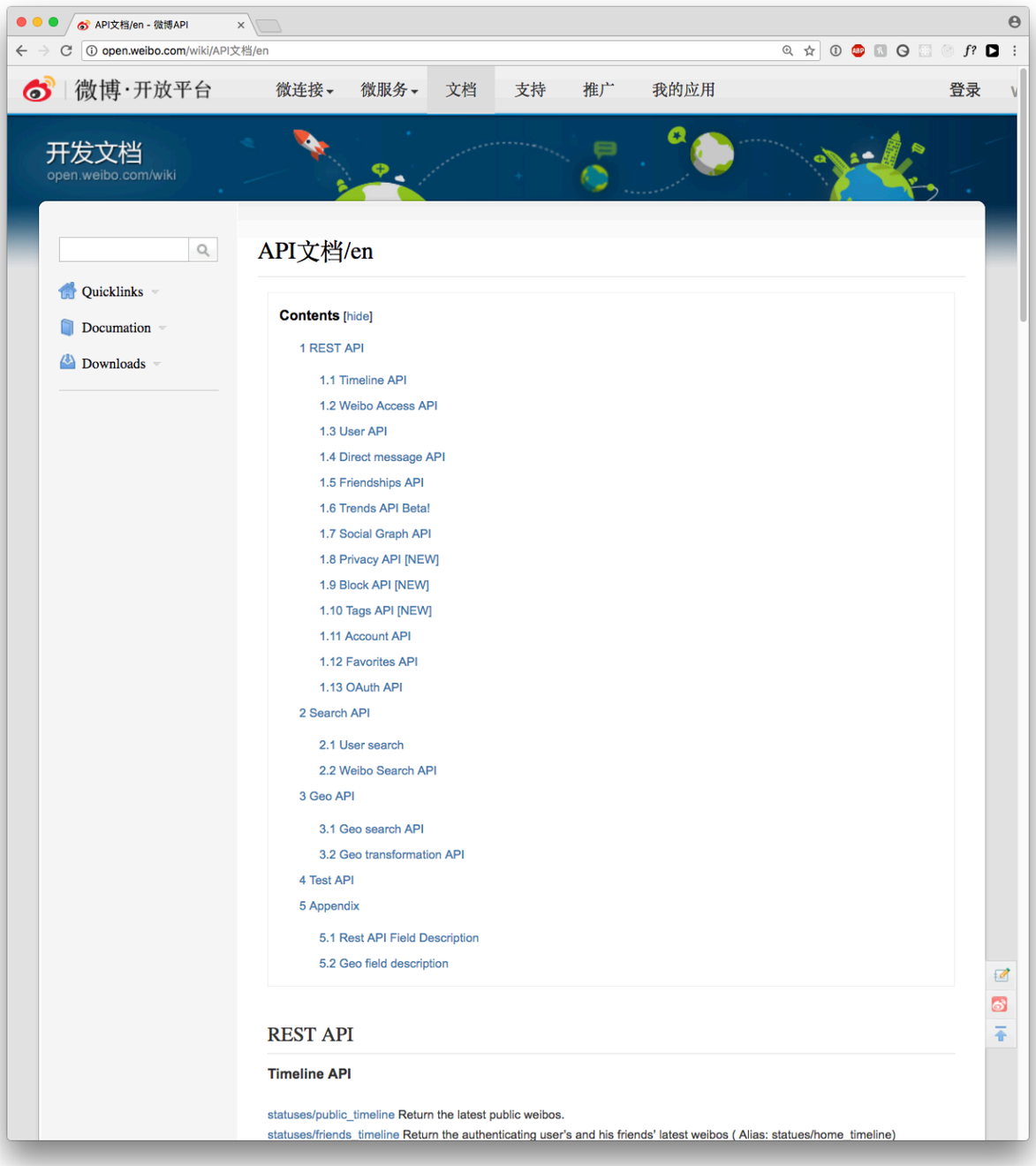
1. Is it a programming language?
2. What is the focus? What does the *domain expert* describe?
3. What is easy, difficult, impossible in this language?

   (relative to a general-purpose programming language)

Example: REST API

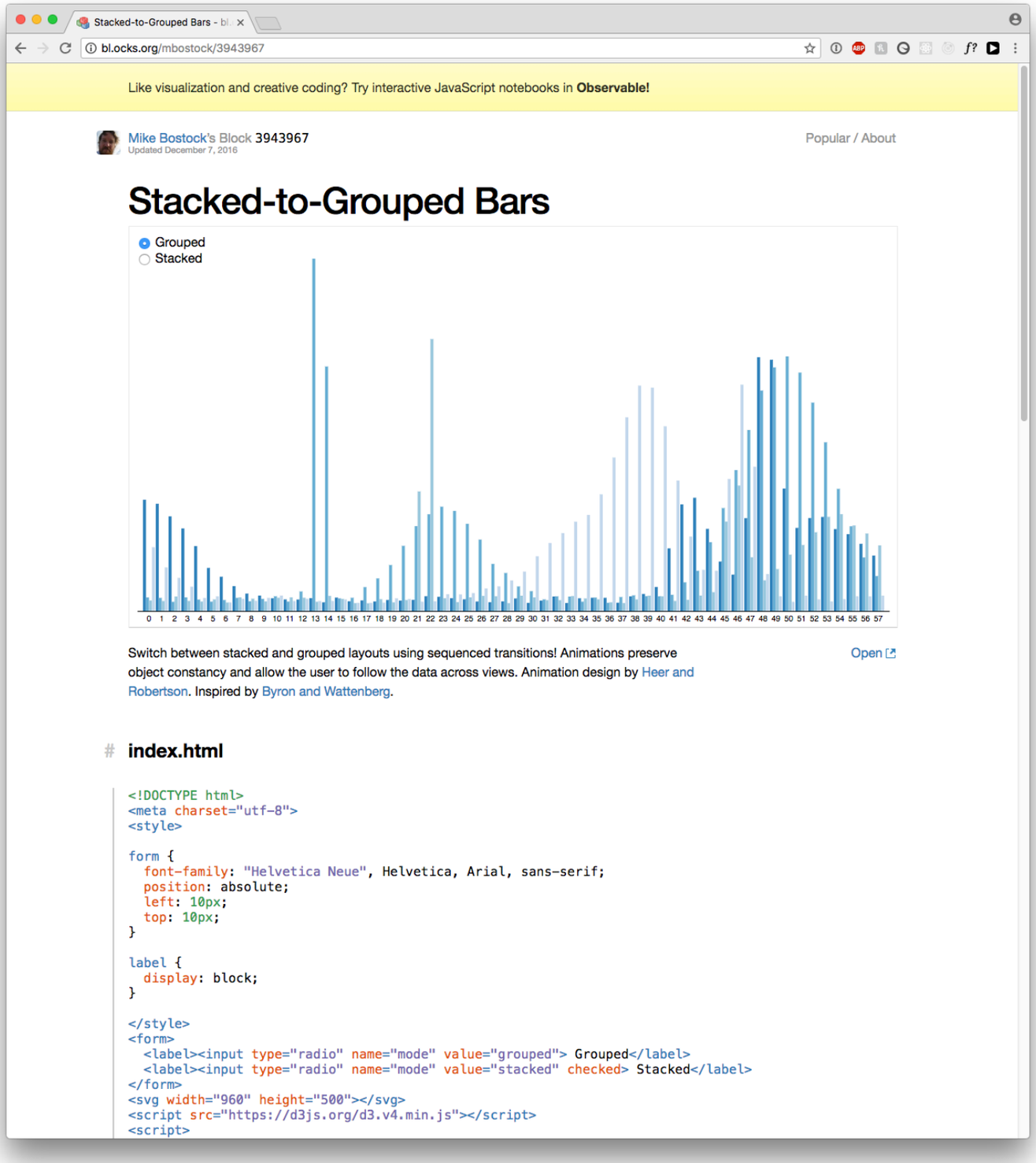


twitter



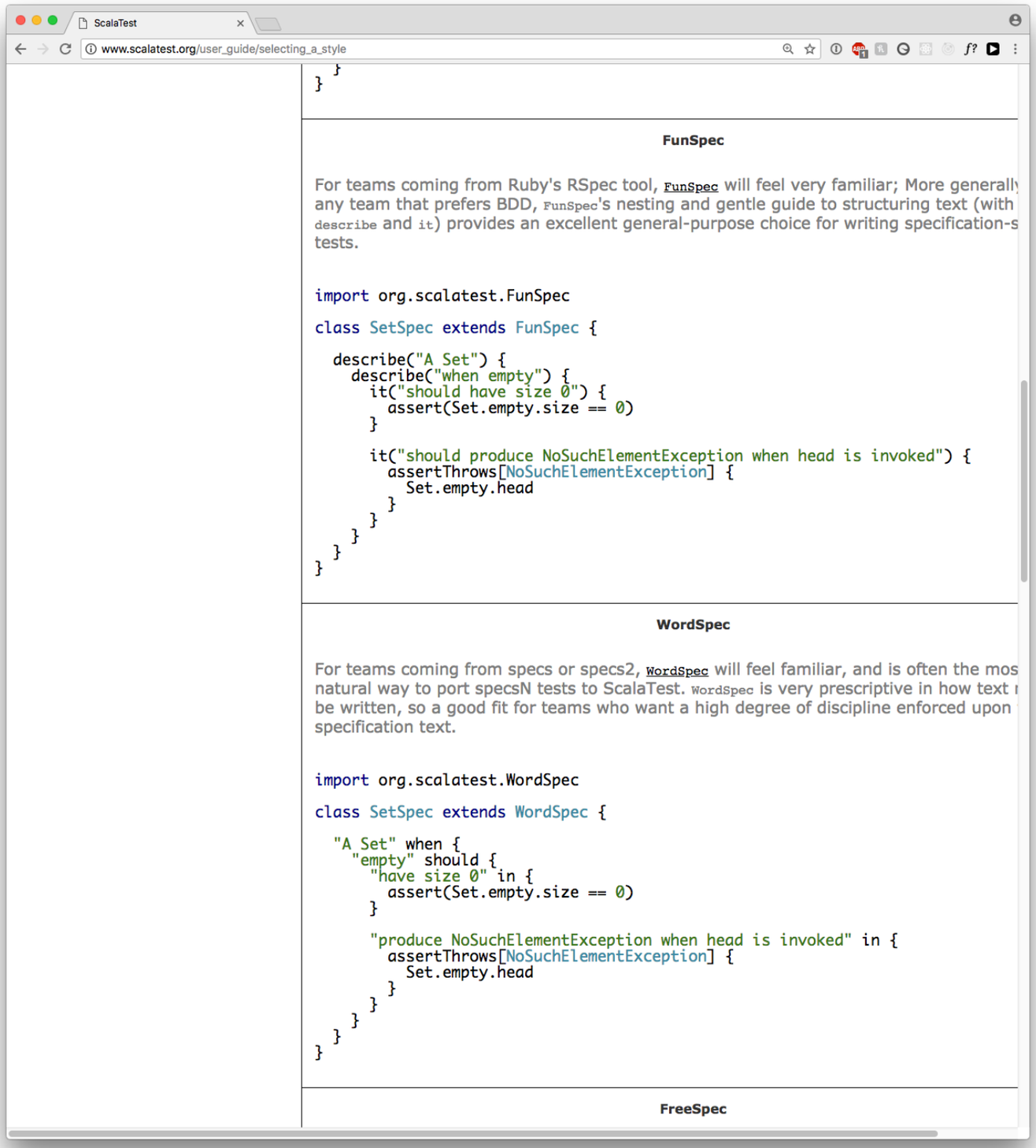
weibo

Example: internal DSL w/ chaining



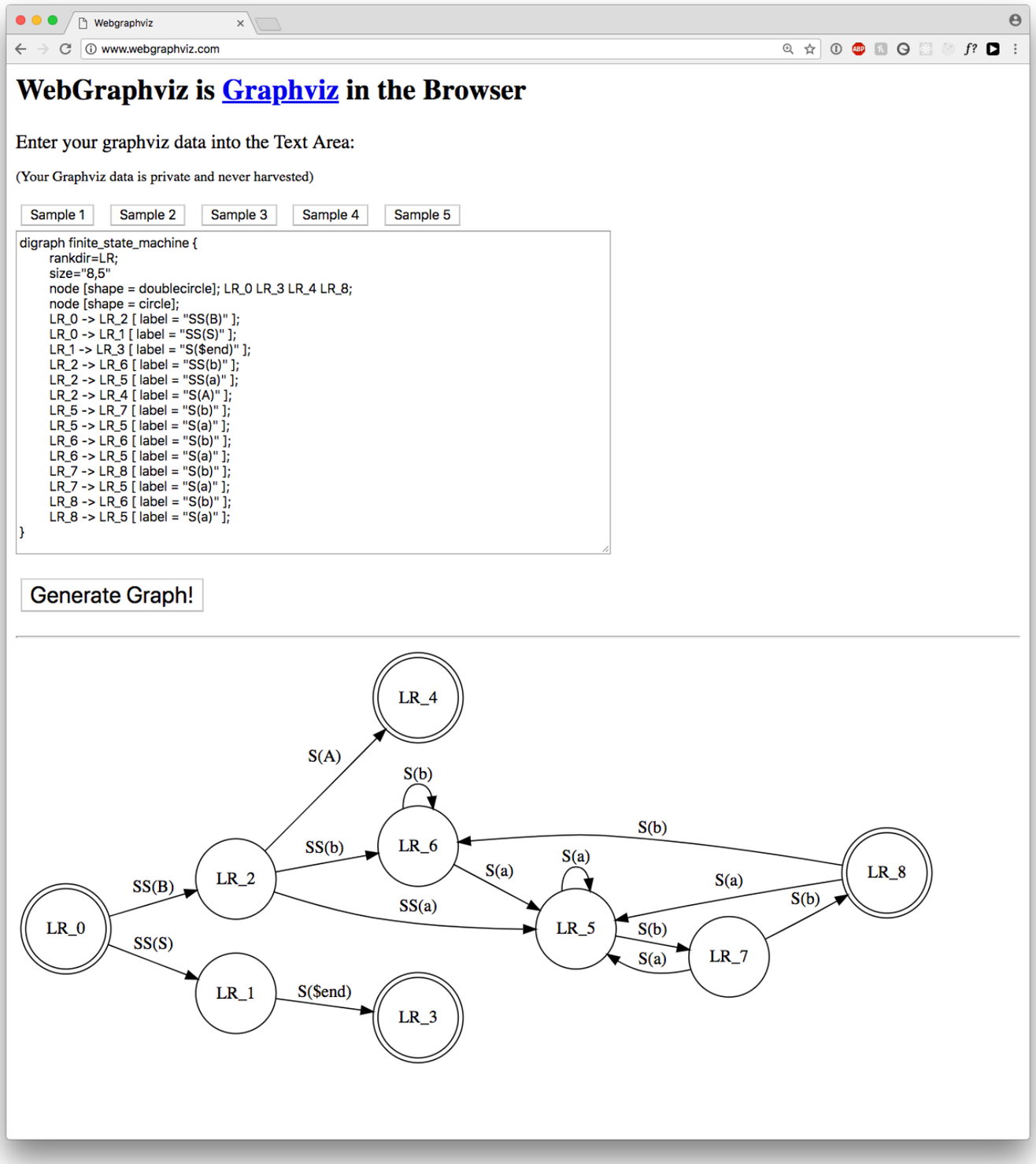
D3

Example: fluent internal DSL



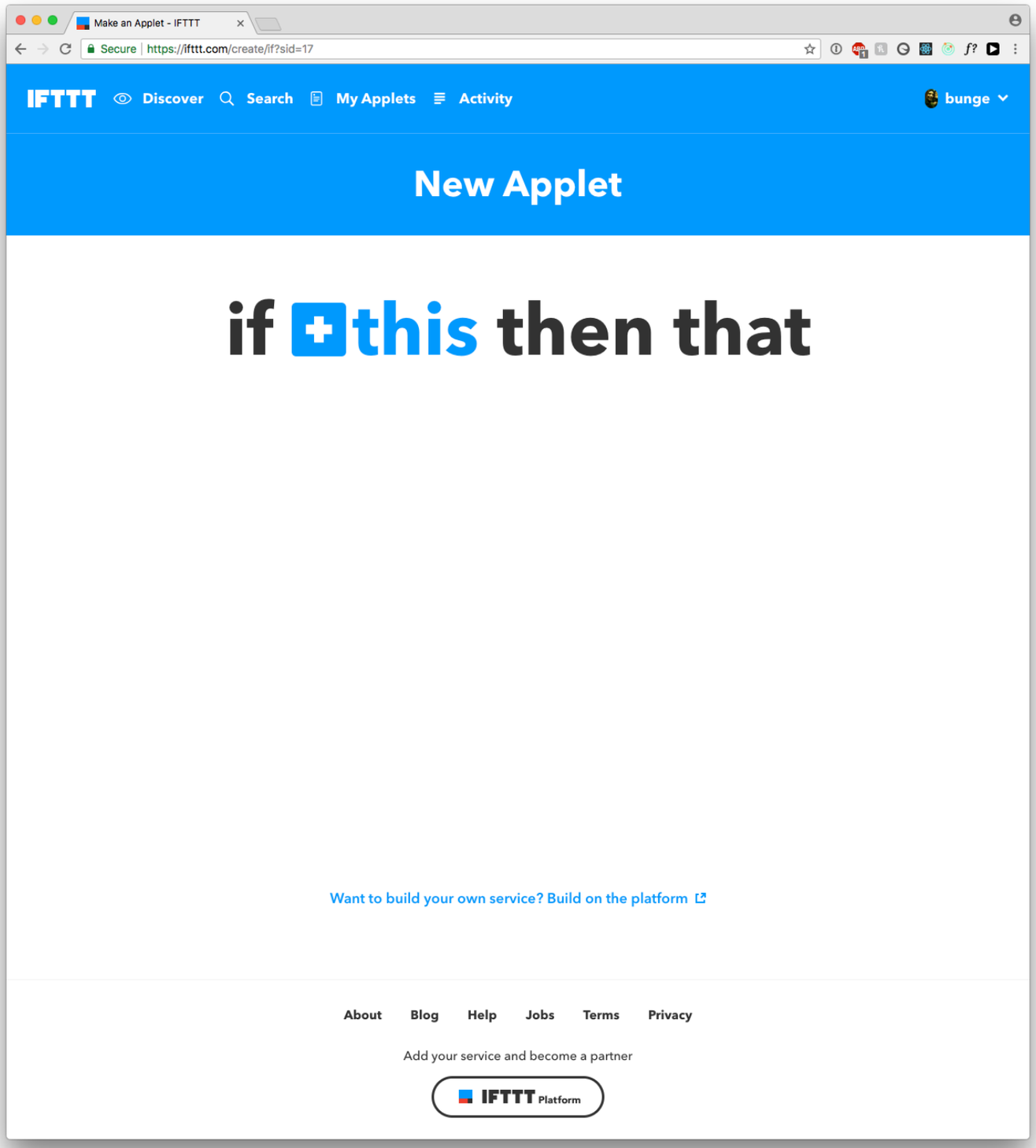
Scalatest

Example: external DSL



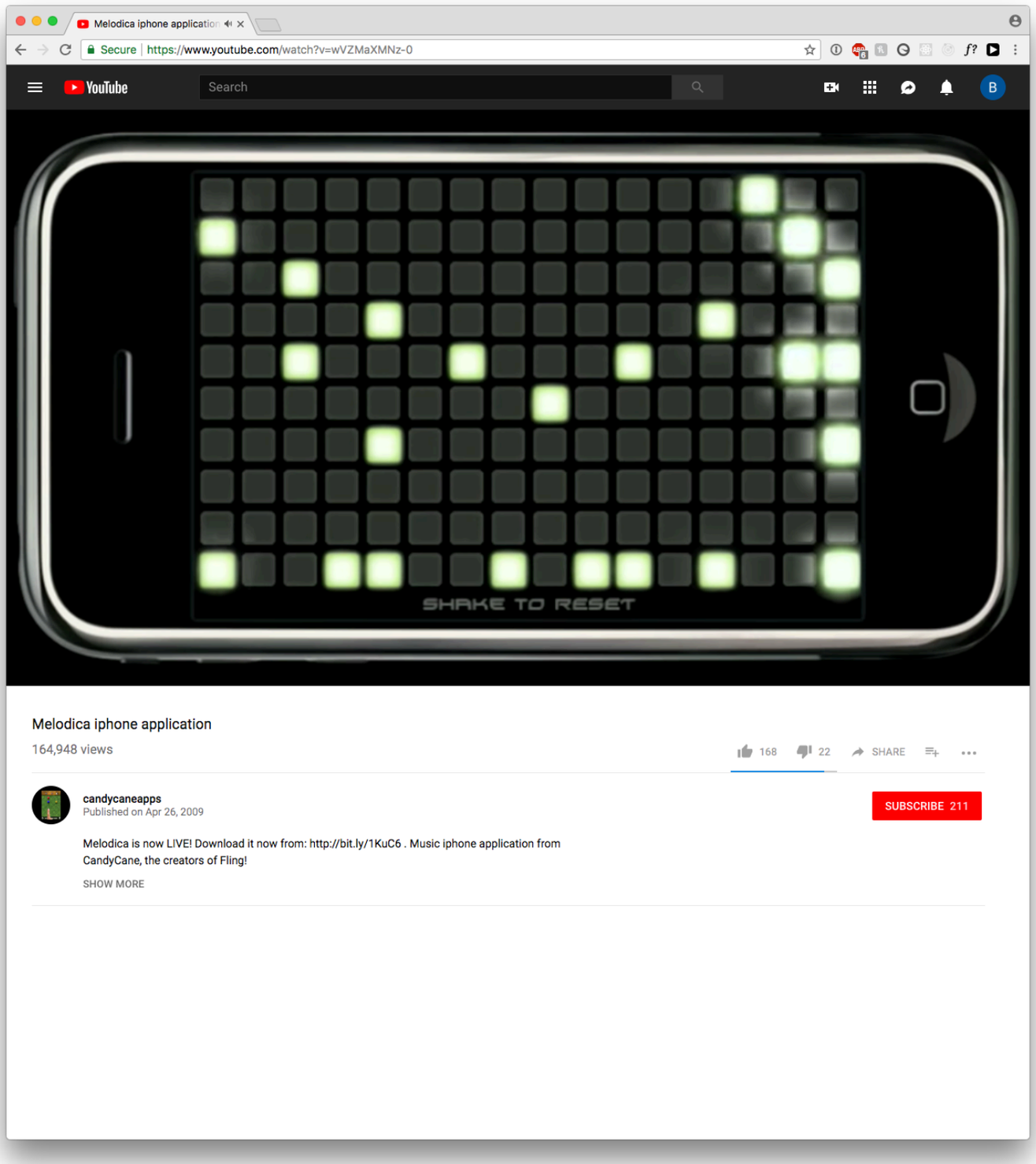
Graphviz

Example



If This Then That (IFTTT)

Example



Melodica