twentytab

# Full-Text Search in Django with PostgreSQL

[EuroPython 2017](#) - Rimini, 2017-07-12

© Paolo Melchiorre - @pauloxnet

# Paolo Melchiorre |

- **Computer Science Engineer**
- **Backend Python Developer** *(>10yrs)*
- **Django Developer** *(~5yrs)*
- **Senior Software Engineer @ 20Tab**
- **Happy Remote Worker**
- **PostgreSQL user, not a DBA**

twentytab

# Goal

**“To show how we have used Django Full-Text Search and PostgreSQL in a Real Project”**

twentytab

# Motivation

**“To implement Full-Text Search using only Django and PostgreSQL functionalities, without resorting to external tools.”**

twentytab

# Agenda

- Full-Text Search
- Existing Solutions
- PostgreSQL Full-Text Search
- Django Full-Text Search Support
- www.concertiaroma.com project
- What’s next
- Conclusions
- Questions

# Full-Text Search |

*"… **Full-Text Search*** refers to techniques for **Searching** a single computer-stored **Document** or a **Collection** in a **Full-Text Database** …"*

**--** Wikipedia

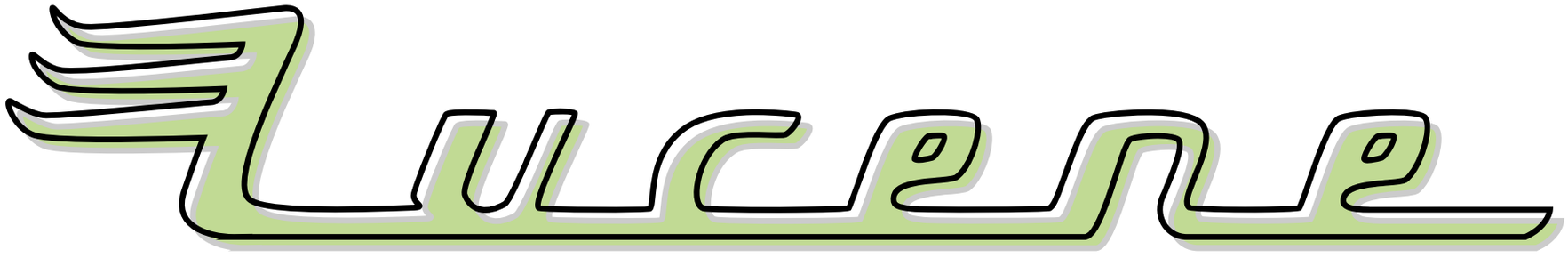
* **FTS** = **F**ull-**T**ext **S**earch

# Features of a FTS

- Stemming
- Ranking
- Stop-words
- Multiple languages support
- Accent support
- Indexing
- Phrase search

twentytab

# Tested Solutions




elastic

Solr

twentytab

# Elasticsearch

Project: **Snap Market** *(~500k mobile users)*

Issues:

- **Management problems**
- **Patching** a **Java plug-in**

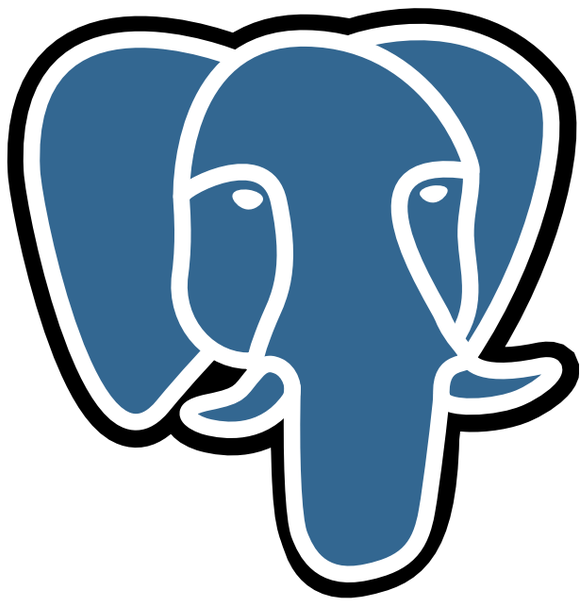
```
@@ -52,7 +52,8 @@ public class DecompoundTokenFilter … {
-               posIncAtt.setPositionIncrement(0);
+               if (!subwordsonly)
+                   posIncAtt.setPositionIncrement(0);
                return true;
            }
```

# Apache Solr

Project: **GoalScout** *(~25k videos)*

Issues:

- **Synchronization problems**
- All **writes** to **PostgreSQL** and **reads** from **Solr**




# Existing Solutions |

**PROS** 👍

- **Full featured solutions**
- **Resources** *(documentations, articles, …)*

**CONS** 👎

- **Synchronization**
- **Mandatory use** of **driver** *(haystack, bungiesearch…)*
- **Ops Oriented: focus on system integrations**

# FTS in PostgreSQL |

- **FTS Support** *since* **version 8.3** *(~2008)*
- **TSVECTOR** to represent **text data**
- **TSQUERY** to represent **search predicates**
- **Special Indexes** (**GIN, GIST**)
- **Phrase Search** *since* **version 9.6** *(~2016)*

twentytab

# What are Documents |

*"… a **Document** is the **Unit** of searching in a **Full-Text Search** system; for example, a magazine **Article** or email **Message** …"*

**-- PostgreSQL documentation**

# Django Support

- **Module:** [django.contrib.postgres](#)
- **FTS Support** *since* **version 1.10** *(2016)*
- **BRIN** and **GIN indexes** *since* **version 1.11** *(2017)*
- **Dev Oriented**: focus on **programming**

# Making queries

```
class Blog(models.Model):
    name = models.CharField(max_length=100)
    tagline = models.TextField()

class Author(models.Model):
    name = models.CharField(max_length=200)
    email = models.EmailField()

class Entry(models.Model):
    blog = models.ForeignKey(Blog)
    headline = models.CharField(max_length=255)
    body_text = models.TextField()
    pub_date = models.DateField()
    authors = models.ManyToManyField(Author)
```

twentytab

# Standard queries

```
>>> Author.objects.filter(name__contains='Terry')
[<Author: Terry Gilliam>, <Author: Terry Jones>]

>>> Author.objects.filter(name__icontains='Erry')
[<Author: Terry Gilliam>, <Author: Terry Jones>,
<Author: Jerry Lewis>]
```

# Unaccented query |

```
>>> from django.contrib.postgres.operations import UnaccentExtension
>>> UnaccentExtension()
>>> Author.objects.filter(name__unaccent__icontains='Hélène')
[<Author: Helen Mirren>, <Author: Helena Bonham Carter>, <Author:
Hélène Joy>]
```

# Trigram similar

```
>>> from django.contrib.postgres.operations import TrigramExtension
>>> TrigramExtension()
>>> Author.objects.filter(name__unaccent__trigram_similar='Hélèn')
[<Author: Helen Mirren>, <Author: Helena Bonham Carter>,
<Author: Hélène Joy>]
```

# The search lookup |

```
>>> Entry.objects.filter(body_text__search='Cheese')
[<Entry: Cheese on Toast recipes>, <Entry: Pizza Recipes>]
```

twentytab

# SearchVector

```
>>> from django.contrib.postgres.search import SearchVector
>>> Entry.objects.annotate(
...     search=SearchVector('body_text', 'blog__tagline'),
... ).filter(search='Cheese')
[<Entry: Cheese on Toast recipes>, <Entry: Pizza Recipes>]
```



twentytab

# SearchQuery

```
>>> from django.contrib.postgres.search import SearchQuery
>>> SearchQuery('potato') & SearchQuery('ireland')
    # potato AND ireland
>>> SearchQuery('potato') | SearchQuery('penguin')
    # potato OR penguin
>>> ~SearchQuery('sausage')
    # NOT sausage
```

# SearchRank

```
>>> from django.contrib.postgres.search import (
...     SearchQuery, SearchRank, SearchVector
... )
>>> vector = SearchVector('body_text')
>>> query = SearchQuery('cheese')
>>> Entry.objects.annotate(
...     rank=SearchRank(vector, query)
... ).order_by('-rank')
[<Entry: Cheese on Toast recipes>, <Entry: Pizza recipes>]
```

twentytab

# Search configuration |

```
>>> from django.contrib.postgres.search import (
...     SearchQuery, SearchVector
... )
>>> Entry.objects.annotate(
...     search=SearchVector('body_text', config='french'),
... ).filter(search=SearchQuery('œuf', config='french'))
[<Entry: Pain perdu>]

>>> from django.db.models import F
>>> Entry.objects.annotate(
...     search=SearchVector('body_text', config=F('blog__lang')),
... ).filter(search=SearchQuery('œuf', config=F('blog__lang')))
[<Entry: Pain perdu>]
```



twentytab

# Weighting queries |

```
>>> from django.contrib.postgres.search import (
...     SearchQuery, SearchRank, SearchVector
... )
>>> vector = SearchVector('body_text', weight='A') +
...           SearchVector('blog__tagline', weight='B')
>>> query = SearchQuery('cheese')
>>> Entry.objects.annotate(
...     rank=SearchRank(vector, query)
... ).filter(rank__gte=0.3).order_by('rank')
```

# SearchVectorField

```
>>> Entry.objects.update(
...     search_vector=SearchVector('body_text')
... )
>>> Entry.objects.filter(search_vector='cheese')
[<Entry: Cheese on Toast recipes>, <Entry: Pizza recipes>]
```

twentytab

# www.concertiaroma.com|

*“… today's shows in the Capital” **

The **numbers** of the project:

~ 1k **venues**

> 12k **bands**

> 15k **shows**

~ 200 **festivals**

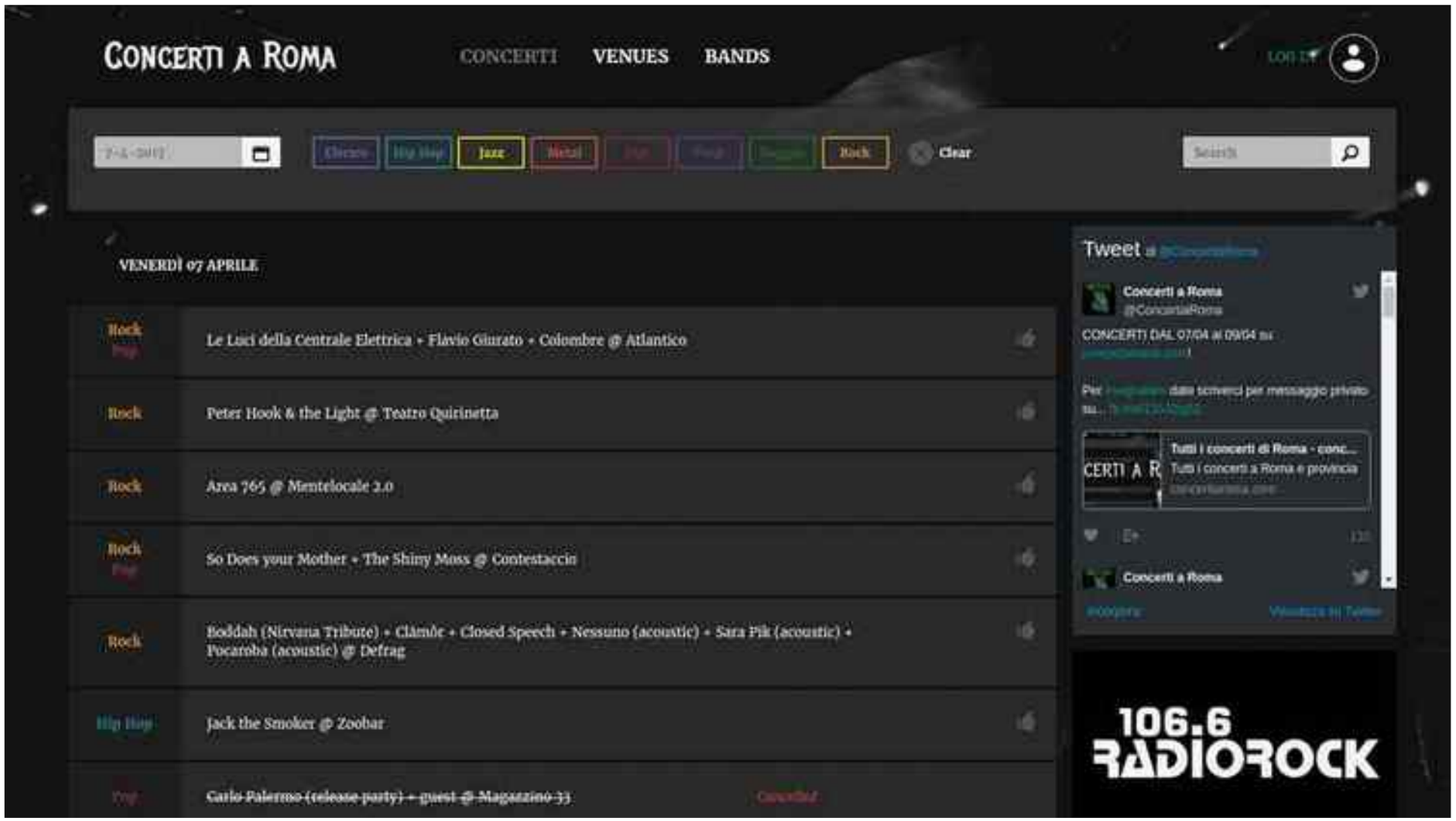
~ 30k **user/month**

* *since ~2014*

twentytab

# Version 2.0

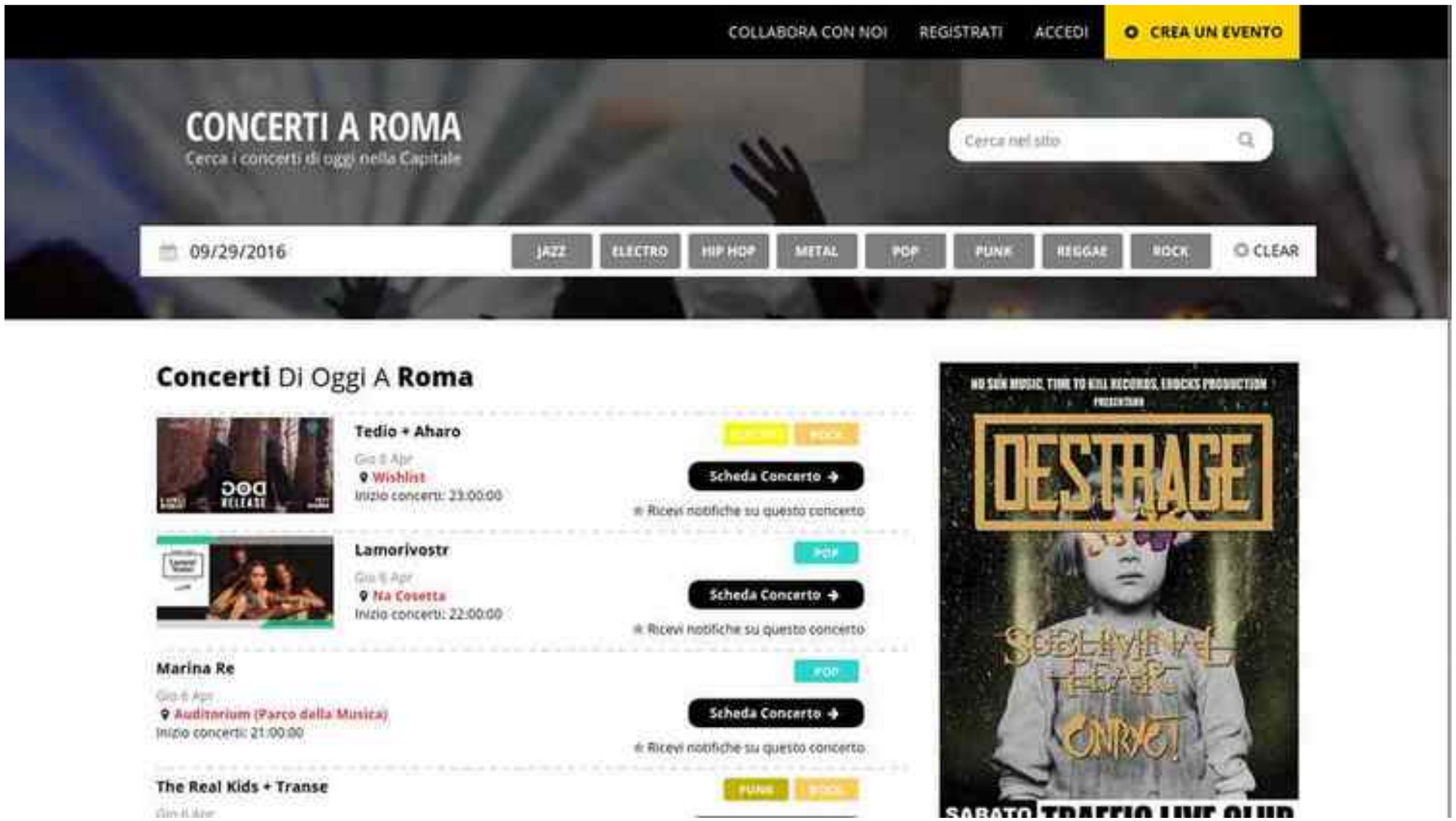
Python **2.7** - Django **1.7** - PostgreSQL **9.1** - SQL **`LIKE`**



twentytab

# Version 3.0

Python **3.6** - Django **1.11** - PostgreSQL **9.6** - PG **FTS**



twentytab

# Band Manager

```
LANG = 'english'
class BandManager(models.Manager):
  def search(self, text):
    vector = (
      SearchVector('nickname', weight='A', config=LANG) +
      SearchVector('genres__name', weight='B', config=LANG)+
      SearchVector('description', weight='D', config=LANG)
    )
    query = SearchQuery(text, config=LANG)
    rate = SearchRank(vector, query)
    return self.get_queryset().annotate(rate=rate).filter(
      search=query).annotate(search=vector).distinct(
      'id', 'rate').order_by('-rate', 'id')
```

# Band Test Setup

```
class BandTest(TestCase):
    def setUp(self):
        metal, _ = Genre.objects.get_or_create(name='Metal')
        doom, _ = Genre.objects.get_or_create(name='Doom')
doomraiser, _ = Contact.objects.get_or_create(
    nickname='Doom raiser', description='Lorem…')
doomraiser.genres.add(doom)
forgotten_tomb, _ = Contact.objects.get_or_create(
    nickname='Forgotten Tomb', description='Lorem…')
forgotten_tomb.genres.add(doom)
....
```

twentytab

# Band Test Method

```
class BandTest(TestCase):
    def setUp(self):
        ...
    def test_band_search(self):
        band_queryset = Band.objects.search(
            'doom').values_list('nickname', 'rate')
        band_list = [
            ('Doom raiser', 0.675475),
            ('The Foreshadowin', 0.258369),
            ('Forgotten Tomb', 0.243171)]
        self.assertSequenceEqual(
            list(OrderedDict(band_queryset).items()),
            band_list)
```



twentytab

# What's next

- **Misspelling** support
- **Multiple** language configuration
- Search **suggestions**
- **SearchVectorField** with **triggers**
- **JSON/JSONB** Full-Text Search
- **RUM** indexing

# Conclusions

Conditions to implement **this** solution:

- **No** extra **dependencies**
- **Not** too **complex** searches
- **Easy** management
- **No** need to **synchronize** data
- **PostgreSQL** already **in** your **stack**
- **Python-only** environment

twentytab

# Resources

- postgresql.org/docs/9.6/static/textsearch.html
- github.com/damoti/django-tsvector-field
- en.wikipedia.org/wiki/Full-text_search
- docs.djangoproject.com/en/1.11/ref/contrib/postgres
- PostgreSQL & Django **source codes**
- **Stack Overflow**
- **Google** ;-)

twentytab

# Acknowledgements

twentytab

Marc Tamlyn

*for **all** the **Support***

*for **django.contrib.postgres***

twentytab

# Thank you



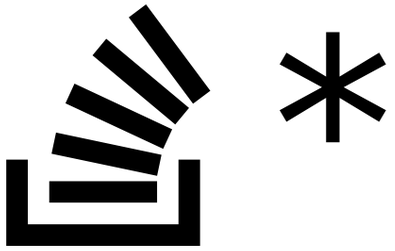
**Slides**

speakerdeck.com/pauloxnet

twentytab

# Questions ?

**After the talk, Please!**



* *Speak **Slowly***

*I'm **not** a native **English** speaker*

# Contacts

www.paulox.net

twitter.com/pauloxnet

linkedin.com/in/paolomelchiorre

github.com/pauloxnet