Approaching Technical Challenges from a **Startup's** Perspective

David Litwin

david@cinely.com

Outline

- Overview of website and business
  - Brief site tour
- Framework for prioritizing tasks
  - Specific challenges faced in Python/Django and how they were dealt with
  - Conceptually-driven
- Wrap-up
  - Why Python? What is next…

Intro

- CINELY is a website to connect and organize the entire production community.
  - It enables filmmakers, actors, and the entire production community to be able to connect with each other, share their work, and find jobs.
- My background and how I arrived at this problem/product.

- S.G.

How it does that

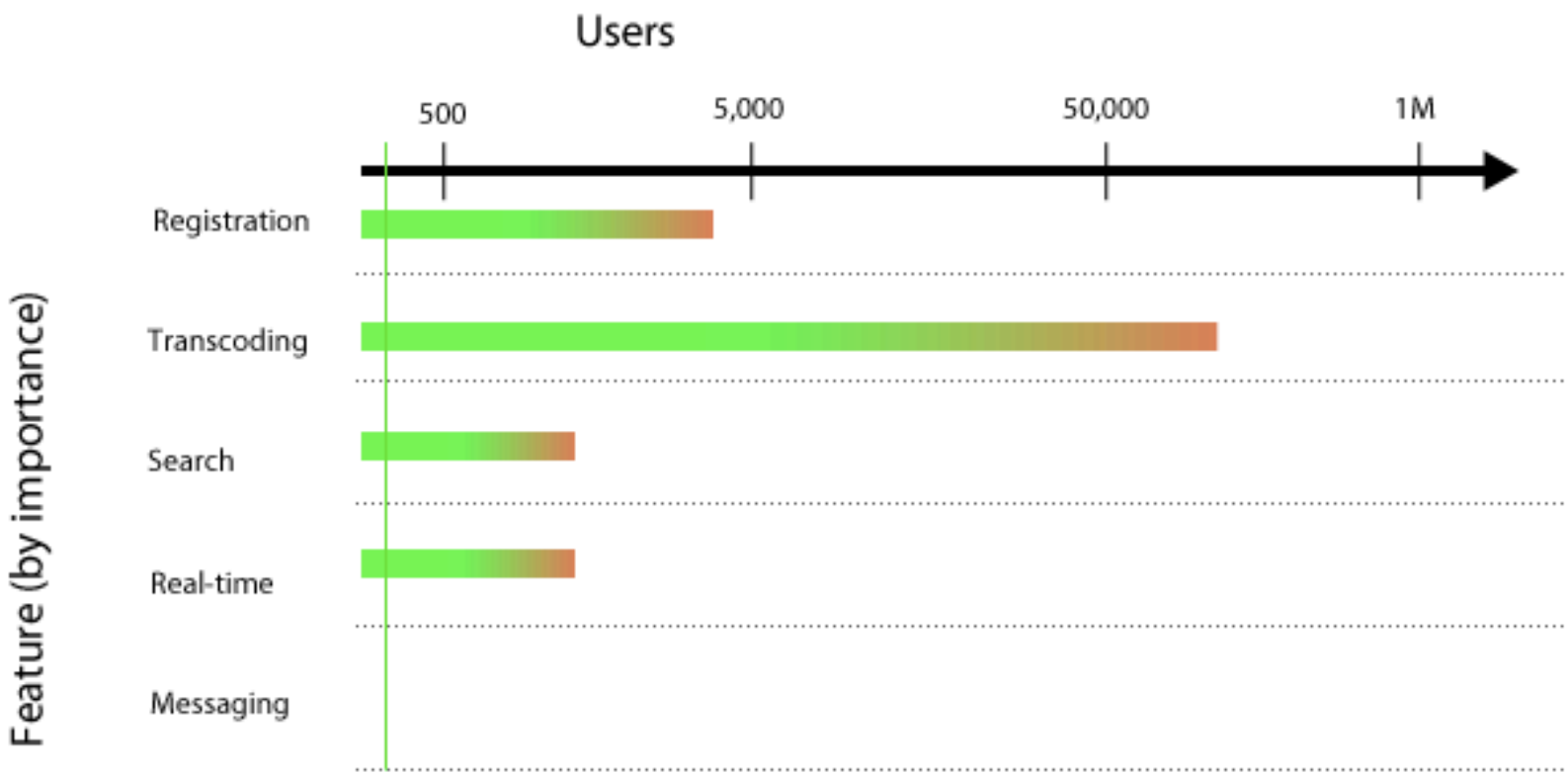
- Quick site tour: www.cinely.com
  - In Private Testing
  - Technology: Python/Django + EC2/S3/CF + Apache + MySQL + Memcached + Github
  - Lines of Code: 10k Python/Django, 1k Unit Tests, 1k JS, 3.5k CSS
  - Upload/share videos, Live Feed, Search, credits

Some of the technical challenges to get here…

- What to do for **Video Transcoding**?
  - ffmpeg, queueing, servers, delivery
- What to do about **Search**?
  - Haystack, Solr, Sphinx, Google?
- What to do for **Real-Time** feeds?
  - Tornado, MySQL triggers?
- Unacceptably **slow ORM** queries

‘Confidence’ interval

- When human capital/time are the primary limitation, the feature only needs to be good enough to get you to point X.



First, prioritze the features.

Second, determine what is acceptable behavior for the application.

Third, define the minimum necessary features and technology
to give an acceptable confidence interval.

Evaluate each option by man-hours and CI. Make a decision and start executing.

Transcoding

At the core of the site.

1. Feature is a priority
2. Acceptable behavior
   - Fast & flawless.
   - Time from upload to live < 1 min per min video
3. Examine approaches. Given an acceptable confidence interval, man-hours are deciding factor.
   - Ffmpeg + ec2
   - API + S3 + CF
4. Determine API + start writing transcoding script

```
88  def process_video(video):
89      try:
90          dimensions = ((width, height), length) = get_video_details(video.filename)
91
92          # store the length into human-friendly format
93          video.length = length.total_seconds()
94
95          # ge aspect ratio
96          video.dimensions = str(width) + 'x' + str(height)
97          print '>>>>>', video.dimensions
98
99          job = api_request_router(video.filename, video_details=dimensions)
100
101         # don't change the status if the response didn't get through
102         if job.code == 201:
103             video.zencoder_id = job.body['id']
104             video.transcoding_status = 'processing'
105             print ">"*20, 'JOB CREATED >> ', job.body['id']
106         else:
107             # mark it as pending for the cron job to retry
108             video.transcoding_status = 'pending'
109             print "Job not created; the cron will retry in a few moments"
110         video.save()
111     except:
112         video.transcoding_status = 'failed'
113         video.save()
114         try:
115             process_video_error(video.filename)
116         except: pass
117         raise
118
```

```
301 def prepare_transcoding_request(url, prefix, options, make_thumbs=False):
302     request = dict(options)
303 
304     filename = url.split('/')[-1]
305     output_url = 's3://' + CONVERTED_BUCKET + '/videos/' + prefix + filename.split('.')[0] + '.mp4'
306     output_url = output_url.strip()
307 
308     request['url'] = output_url
309 
310     request['notifications'][0]['url'] = "{0}transcoding/zencoder_response/".format(settings.URL_ROOT)
311 
312     if make_thumbs:
313         filename_clean = clean_filename(filename)
314         request['thumbnails'][0]['filename'] = filename_clean + '_large'
315         request['thumbnails'][1]['filename'] = filename_clean + '_thumb'
316         request['thumbnails'][2]['filename'] = filename_clean + '_mini'
317         request['thumbnails'][3]['filename'] = filename_clean + '_recentactivity'
318         expire_date = datetime.now() + timedelta(days=365)
319         formatted_time = format_date_time(mktime(expire_date.timetuple()))
320 
321         for thumb_num in range(4):
322             request['thumbnails'][thumb_num]['headers'] = {
323                 'Expires': formatted_time
324             }
325 
326     return request
327 
328 
329 def create_transcoding_job(url, outputs):
330     zen = Zencoder(settings.ZENCODER_API_KEY)
331     job=zen.job.create(zencoder_quote(url), outputs=outputs)
332 
333     return job
```

Real-time feed

- Does it need to be live?
- “The original version of News Feed on Facebook, which was up for over two years before being replaced, actually wasn’t real time. We swept through all the users about every 15 or 30 minutes and wrote the new stories to MySQL in large updates. We would publish stories at 5 minute intervals into the future to give the appearance of a steady stream of information.”
  - Andrew Bosworth, creator of News Feed

Real-time feed

- Signals + AJAX long-polling + Memcached

Pseudocode:

```
def recent_activity_post_save():
    notify_view()

[in the view]
while not new_activity():
    sleep(1)
return HttpResponse(new_activity())
```

Search

- Solr is good, Haystack is good, Sphinx is good, Google is good.
- Tried Sphinx.
- What about LIKE?

```
216     ### for SEARCH - chain queryset after filter, so if a filter has been applied
217     ### it will limit search to those filtered results
218     if search:
219         search = request.session['search'] = search.strip()
220         # double percent signs are needed because django needs them escaped
221         # we also make have to manually take care of SQL injection risks by removing quotes
222         like_query = '%%' + '%%'.join(search.replace('"', '').split()) + '%%'
223         video_set = video_set.extra(where=['slug LIKE "{0}"'.format(like_query)])
```

Django's ORM – A great start

- Good and easy for most cases.
  - Simple writes
  - Reads OK with .filter, .exclude, .get ~95% of time.
- Sometimes generates painfully slow queries.
  - Try using .select_related in debug-toolbar.
- Still too slow or not working?
  - Don't force it (usually will take a lot of time and won't work anyways).
  - Use SQL -- .extra or custom model method

Get video credit count using .extra

```
86      elif sort == 'credits':
87          talent_set = talent_set.extra(select=
88          { 'count':
89              """
90                  SELECT (
91                      SELECT COUNT( DISTINCT v.id )
92                      FROM videos_videocredit c
93                      LEFT JOIN videos_video v ON c.video_id = v.id
94                      WHERE c.profile_id = userprofile_userprofile.id AND v.uploaded_by_id != userprofile_userprofile.id
95                          AND v.active
96                   ) + (
97                      SELECT COUNT(*)
98                      FROM videos_video v
99                      WHERE v.uploaded_by_id = userprofile_userprofile.id
100                         AND v.active
101                  )
102             """ })
103
104         talent = talent_set.order_by('-count')
105     else:
106         talent = talent_set.order_by('user__date_joined')
107     return talent
```

Taking a step back

- Though each feature presents its own challenges, the biggest ‘technical’ challenge for a startup is efficiently allocating effort to the correct task!

1. Knowing exactly what you want…and changing it based upon what users want.
2. Prioritizing effort to the most important task(s).
3. Being precisely clear in instructions.

Wrap-up

- 6 months to develop.
  - Why so long?
- Zero programming experience. Learned Python to build this site.
- Why Python?
  - Large, serious, and helpful community
  - Django has great documentation
- **Looking for a great developer to join the team.**
  - Salary + equity
  - Passion for film a plus
  - Email me at: david@cinely.com