

Connected GameStore

Play! framework showcase

Outline

play! showcase

- Introductions
- Connected GameStore Product
- Technical highlights
- Play! experience
- Future plans



About BoosterMedia

introduction

- Founded 2009 as spin-off of mobile web company
- 'Mobile Gaming only' → focus on mobile social games
- Games & Platform development
- Reach > 2M users/month
- Web platform 12M hits/month
- Traffic increase 25-50%/month
- 30 employees, growing fast



About me

introduction

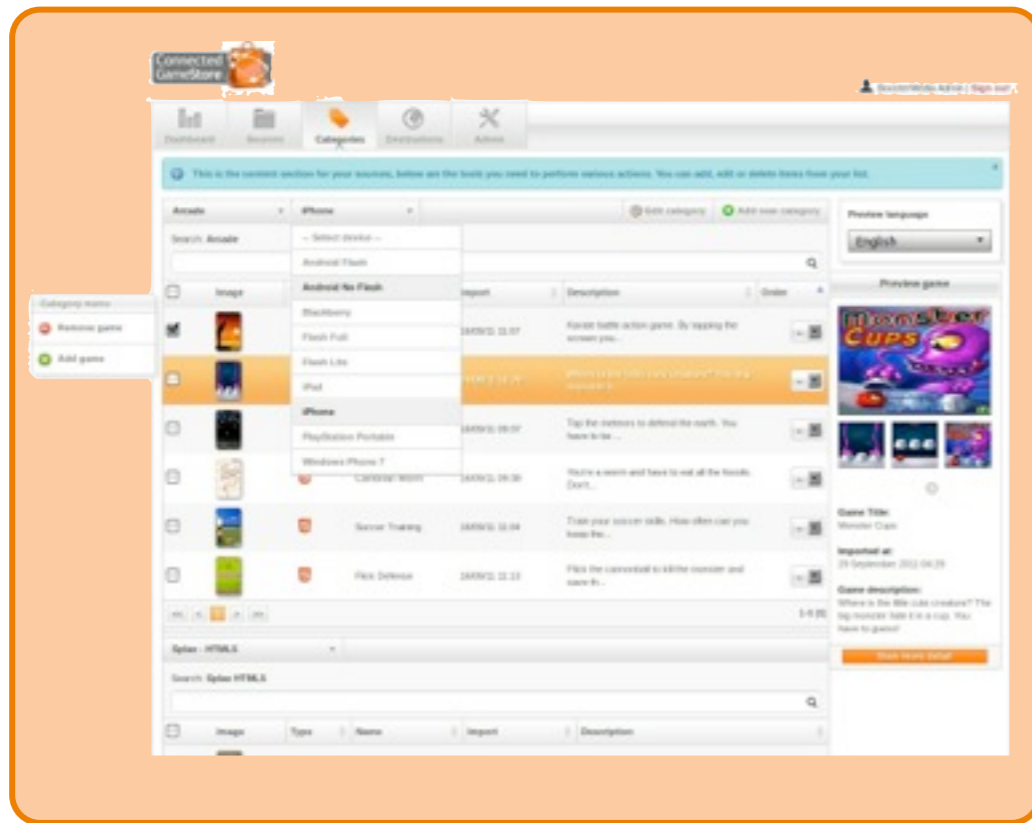
- Electrical Engineering DUT '97
- Web developer (back-end) since '98
- 10 years of perl experience
- 4 years of java experience, SCJP
- Certified scrum master
- Head of Technology BoosterMedia



Product Overview

Connected GameStore

Goal: Bring the right mobile game to the right mobile device



Games



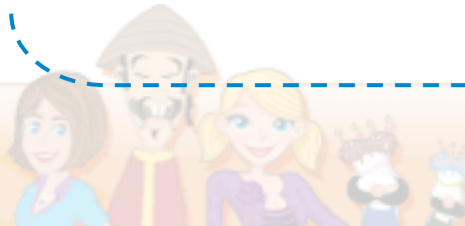
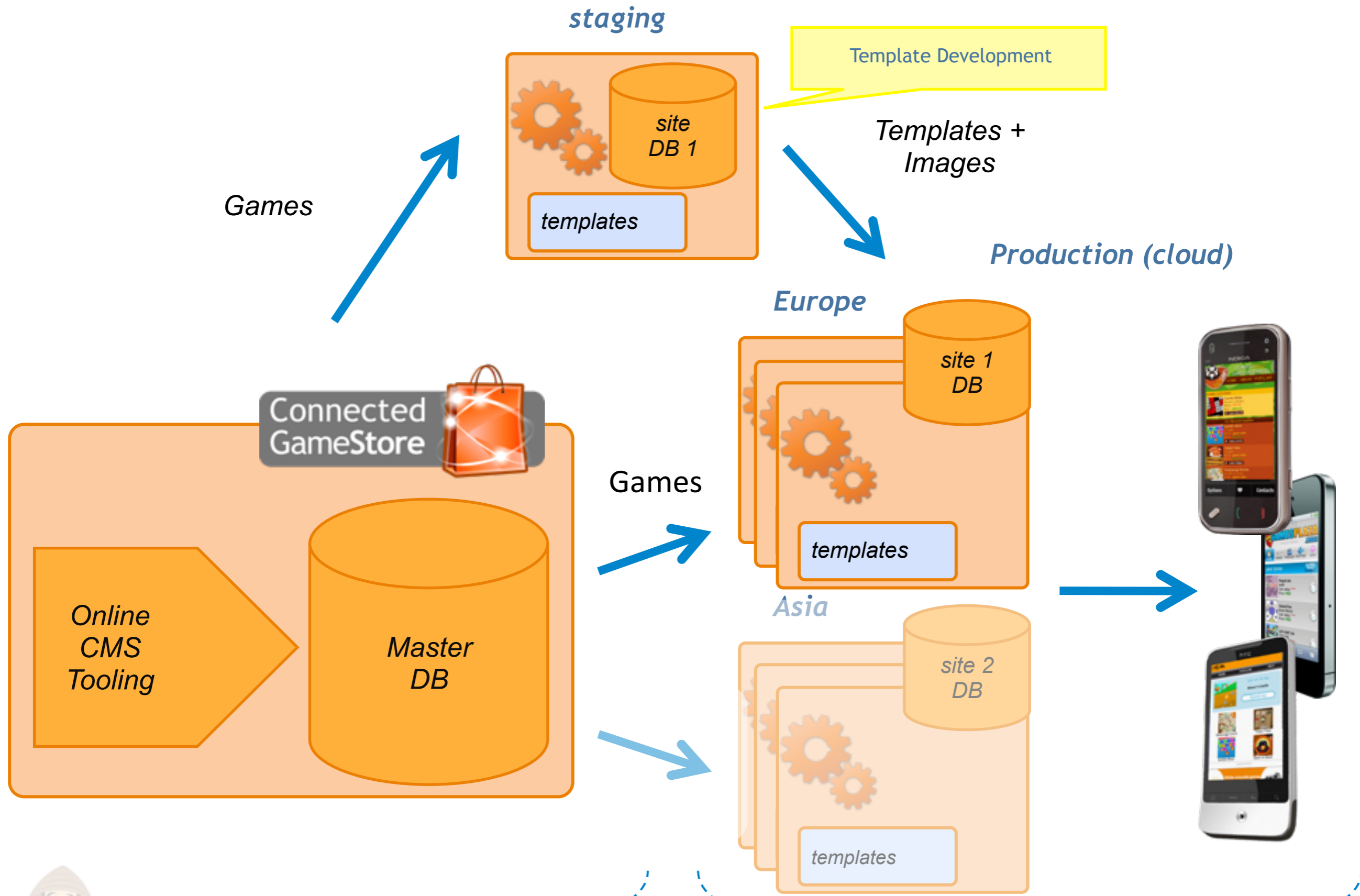
Back-end: CMS

Front-end: Portal



Architecture

Connected GameStore



Back-end

Front-end

Demo

Connected GameStore

screencast



Project

Connected GameStore

- Team 3-4 coders (including PO/SM)
- Time spend 1 year / 5K hours
- Scrum 2 week sprints (3 week cycle)
- 60k lines of code
- Automated tests
 - Unit/Functional tests: 350
 - Selenium test: 45



Technical Highlights

Warning



Pragmatic Approach




Some details...

- Running Play! 1.2.4
- Play JPA / Mysql (explicit foreign keys)
- Multi-tenant "by 'hand'" (virtual host based)
- Image rescaling - IM4Java (imageMagick)
- HTML/CSS/Jquery (datatables/uploadify/jqTools)
- Client/server comm Gson (cms/portal)
- Hazelcast for intra-server communication
- VPS based hosting (EC2)
- Apache FE



- Reason: not comfortable with ‘svn up’
- Build process (ant based)
 - Check out modules + project of a branch/revision
 - Run dependencies on modules + project
 - Pre-compile project
 - Replace module-path by placeholder
 - zip precompiled project, store in svn
- Roll out process
 - checkout and unpack release
 - replace module paths
 - Stop server/switch directories/Start server



- We use Jira Studio  Bamboo for “free”
- Only Elastic Bamboo - EC2 ‘small’ instance
- Process
 - Execute build process (checkout modules, run deps)
 - Patch log4j to use STDERR
 - Remove selenium tests
 - Play ‘auto-test’
- TODO: use webdriver module to use real browser



- Need for controlled db versioning (no evolutions yet)
- Our solution
 - use hand-rolled php script (executes SQL)
 - independent of play! (used in php projects)
 - no down-grading
 - play module
 - application.conf: database.version
 - ‘Evolutions-like’ db version checker
 - play! fixtures wrapper (evolutions ‘cheats’)

Alternative: flyway (?)



■ Portal layout requirements

1. Flexible (*support evolving standards*)
2. Front-end developer friendly (*error messaging*)
3. Minimize (tech) implementation effort
4. Deploy independent from source
5. Put Front End developer in control (*view-first*)

■ Options considered

- Database driven approach
- Use a templating engine

■ Play! groovy templating: 4 out for 5 (only #4 missing)



Implementation

- Custom template loader
 - Select template
 - based on device characteristics
 - fallback to alternative
 - Read templates from outside play! root
 - Hot-reloading in production mode
- Custom tags (= ~ 30)
 - load game categories, games, images
 - conditional content
 - caching
 - etc...



Portal Templates

Technical Highlights

Url : /games/acade/arcade-shoot-m-up

Route: /{pageCode}/{catCode}/{seoName}

```
#{theme.themeExtends 'Page/main.html' /}
```

```
#{categoryIndex codeParam:'catCode', as:'res', pageSize:10 /}
```

```
<p>Games for ${res.cmsCategory?.name}</p>
```

```
<div class="qitem-ad">#{advertisementMocean /}</div>
```

```
<ul>
```

```
  #{list items:res.cmsGames, as:'game' }
```

```
    <li><a href="#{link.game pageCode: 'game', game: game /}">
```

```
      #{gameImage displayType:'square', game:game, width:'77px' /}
```

```
      <h2>${game.name.raw()}</h2>
```

```
    </a>
```

```
  </li>
```

```
#{/list}
```

```
#{else}
```

```
  <li>no games found</li>
```

```
#{/else}
```

```
</ul>
```

```
#{theme.themeInclude 'Page/genre_list_games.html' /}
```



- Issue: play! log4j logger has only 'Play' category
- solution: create our own logger
 - copied string-format behaviour
- Additional features
 - error mailer (modified smtp appender)
 - aggregating & throttling (EC2!)
 - blacklisting (play! logger)
 - enhance message (w/ request.domain)



■ Load tests

- EC2 large (2CPU)
- Amazon RDS (large)
- Jmeter from local (EC2) network
- “real” index page
- mixed user-agents

■ 10 page req/sec = ~ 25M hits/month

■ Optimizations done

- serve static content through apache/mod_perl
- cache: device recognition, template detection
- async loading of advertisements



Play! experience

■ Why we chose play! framework

- Big project with long lifespan → Java seems good choice
- J2EE didn't bring much joy
- Play's focus on productivity (*feature rich*)
- Need for a framework that support agile development
- Build w/ scalability in mind (*shared-nothing*)
- Active community

■ Our experience

- + Developer Experience is great
- + (you can hot-fix your javacode on production)
- ~ We ignore some features `get/set` generation, local args for render
- ~ Attention: namespacing/static use - beware
- async features work, but impact your code a lot



Future Plans

■ Features

- Portal: social integration
- More sophisticated search (Elastic Search)
- Stream line CMS
-

■ Technology

- faster templates (@mbknor)
- DB scalability JPA/MySQL, maybe NoSql
- Play 2.0
- Scala(?)



We're Hiring!



www.boostermedia.com



Thanks!

Questions?

