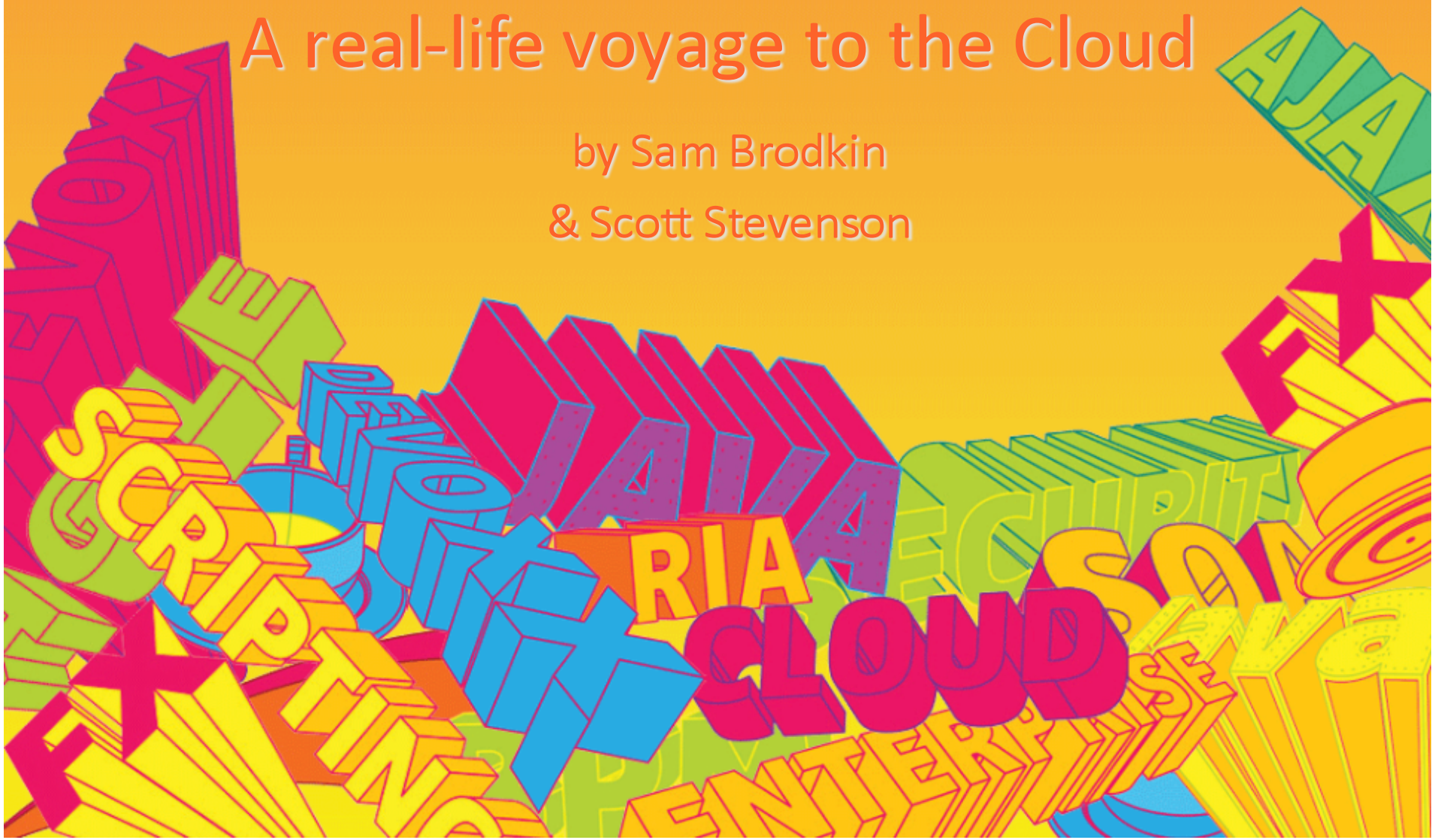
The Google App Engine for Java.
A real-life voyage to the Cloud

by Sam Brodkin

& Scott Stevenson



Speakers

- Sam Brodkin is a freelance Java technologist, former Sun trainer and organizer of the Dutch Google Technology User Group. Member of J-Knights JUG.

- Scott Stevenson is a freelance Java consultant with over 20 years experience in the industry and is a member of the Dutch GTUG and the J-Knights JUG.

Aims of Presentation

- Introduction to the GAE/J
- Share real-life experiences developing a non-trivial application on the App Engine.
- Highlight limitations, technology choices, frustrations and successes.
- Show you how to create an application on App Engine and dig into our code during the presentation.

Agenda

- SwagSwap Demo and installation
- Introduction to Google App Engine
- SwagSwap in-depth
- Create your own App Engine application
- Demonstrate production environment

Participation

5 possible levels of participation during this talk:

- **Zombie** - Just sit there for three hours
- **Bug patrol** - Find and log bugs in SwagSwap
- **Lurker** - download and run the code but don't contribute
- **Inquisitor** - Ask questions interactively or using Moderator
- **Contributor** - add a feature or fix a bug during the presentation

Get Involved

- Set up a Google account
- Download SwagSwap code
- Download App Engine Plugin for Eclipse
- Use Google Moderator to ask questions
- Use SwagSwap application during presentation

SwagSwap Demo

Introduction to Google Application Engine for Java

What's this demo about

What is Google App Engine

- A Cloud computing platform
- Run your apps on Google's infrastructure
- Google Provide the container and services
  - Hardware and connectivity
  - Operating System
  - JVM
  - Servlet Container
  - Software Services

Key Features

- No need to install or maintain your own infrastructure
- Google scales for you
- Charge only for actual usage (with free quota)
- Built-in application management console

Limitations of the App Engine

- Requests limited to 30 seconds
- No long-running background processes
- No Server push
- No threads
- Read-only file system
- Whitelisted classes

Development Environment

- Emulates the production environment
- Customized Jetty server
- Local implementations of MemCache and Disk-backed datastore
- WhiteList for compile-time checks
- Google Eclipse Plugin

Datastore

- Runs on BigTable
  - Distributed
  - Scalable
  - Transactional
  - Schema-less
- NOT a relational database
- It’s an object store
- JDO / JPA Support

Quotas and Billing

Resource	Provided Free	Additional Cost
CPU	6.5 hours/day	$0.10/hour
Bandwidth In	1GByte/day	$0.10/GByte
Bandwidth Out	1GByte/day	$0.12/GByte
Stored Data	1 GB	$0.005/GB-day
Emails sent	2000/day to users 5000/day to admins	$0.0001/email

SwagSwap In Depth

Technology Highlights

- Spring 3.0: JDO, transactions, service layer, RESTful Web MVC
- 3 Front-end Implementations. Spring MVC, JSF 2.0 and GWT 1.7 / SmartGWT.
- Declarative build, test and deploy (ANT)
- Google Code for issues, version control, release management and Wiki.

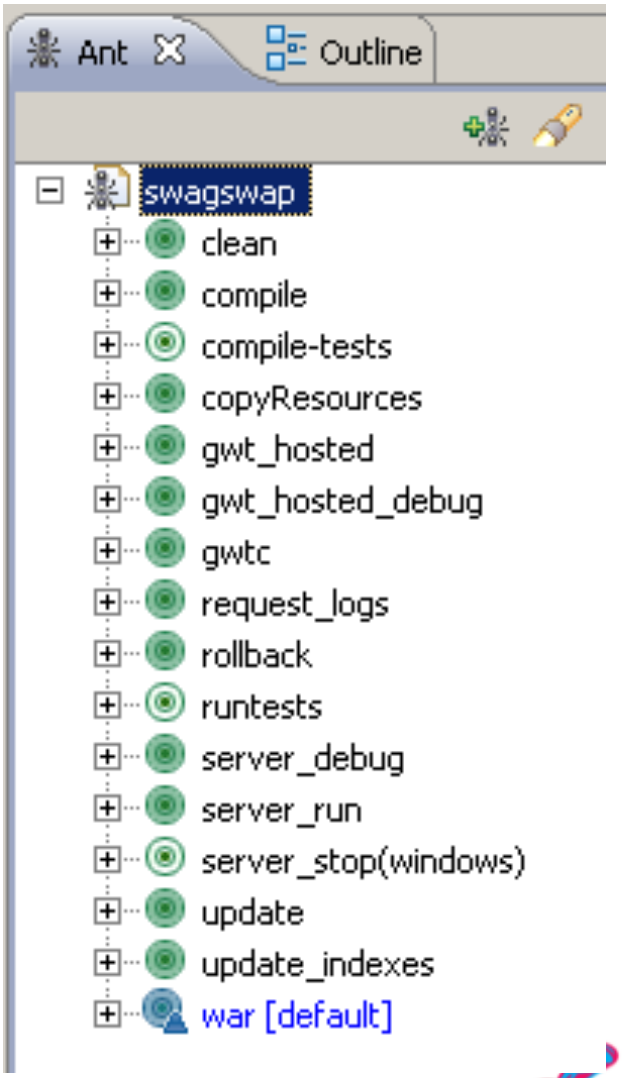
Dev Environment

- Appengine development server
  - Simulates the App Engine Java runtime environment and all of its services, including the datastore and logging in

- GWT hosted mode browser

ANT build.xml

- **`runtests`** – runs the unit tests
- **`server_run`** - runs Appengine dev server
- **`update`** – upload the app to Appengine
  - Make sure the war is built before you do that (run gwtc and war tasks)
- **`rollback`** cleans up any partial failed deployments



SwagSwap - Service Layer

What's this demo about

Architecture

- 3-tiered J2EE
  - JDO (and CacheManager) DAO layer (ala Spring)
  - Service layer (Spring POJOs) for transactionality, authentication, orchestration, and testability
  - Front end - 4 implementations that all use the same service layer

The Spring Config

- Spring MVC stuff including annotations-config
- JDO **`PersistenceManagerFactory`** wired to the appengine datastore
- Spring annotated transactions and transaction manager
- DAOs
- Services (with DAOs and other services injected)
- SwagCacheManager classes

SwagItem (1)

- Core Domain object
- Uses JDO annotations:

```
@Persistent
private String name;
```

- Also uses an Appengine specific one:

```
@PrimaryKey
@Persistent(valueStrategy =
IdGeneratorStrategy.IDENTITY)
private Long key;
```

SwagItem (2)

- Has some Appengine types too
  - Text
  - Blob
- Also has a owned one-to-one relationship:

```
@Persistent
private SwagImage image
```

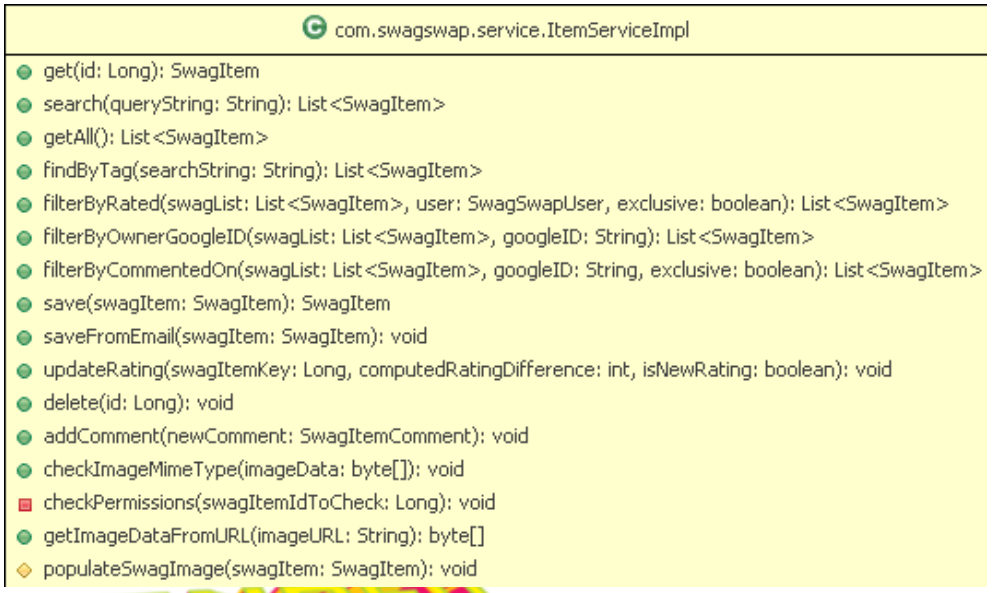
- And an owned one-to-many

```
@Persistent
private List<SwagItemComment>
    comments
```

ItemService Overview

Responsible for:

- exposing CRUD operations on swag items
- Using the SwagSwapUserService for authorization and creating users (users are created when they login)



ItemService.save()

When a swag item is saved (added or updated):

- Check for swag item name (only required attribute)
- Assume they're logged in, get the SwagSwapUser
- Populate swag item with user details (Google ID, nickname)
- Add Image (if any) from URL or file upload
- save (transactionally via spring annotations!)
- If it's an update (!swagItem.isNew()) check permissions (only the owner or admin can update an item)

ItemService. populateSwagImage()

Sets SwagImage in SwagItem (JDO child Object, 1-1 managed relationship)

- First looks for imagebytes from a fileupload
- Then looks for an image URL and pulls it down using Appengine URL Fetch (just java.net.URL openStream())
- Checks the mimetype using jMimeMagic
- Resizes the image using the Appengine Image API
- Throws exceptions for invalid images

ItemDao

- Uses Spring JDOTemplate which
  - Ensures that the persistenceManager is properly opened and closed
  - Automatically participates in transactions
  - Translates DataExceptions to Spring's RuntimeException hierarchy
- Has a synchronized method updateRating()
- Has a search() method but can't search case-insensitively ☹

Login / Signup

We're using Google Accounts for authentication

- On appengine it's as easy as protecting resources in web.xml
- However, it's a weird programming model.
  - It's awkward to create your own User objects
  - You have to decide when to persist them to your DB
  - You don't have your own sign in screen (or signup screen!)
  - Signing into SwagSwap is like adding a Facebook app - The author off the app gets your email address and nickname but not much more

Protecting resources in web.xml

- Trying to access protected resources when not logged in through Google Accounts send you to the Google login page (and redirects back)

```
<security-constraint>
    <web-resource-collection>
        <url-pattern>/springmvc/delete/*</url-pattern>
        <url-pattern>/springmvc/add/*</url-pattern>
        <url-pattern>/springmvc/edit/*</url-pattern>
        <url-pattern>/springmvc/rate/*</url-pattern>
    </web-resource-collection>
        <auth-constraint>
            <role-name>*</role-name>
        </auth-constraint>
    </security-constraint>
<security-constraint>
```


Protected Resource GOTCHA

- When a protect URL is requested by a not-authenticated user
  - The user is sent to the Google login page
  - After logging in, the user is sent back to the requested URL
  - GOTCHA: query string parameters are dropped on redirect
  - Why does that suck?
  - I filed a bug on this which has been accepted (issue 2225)

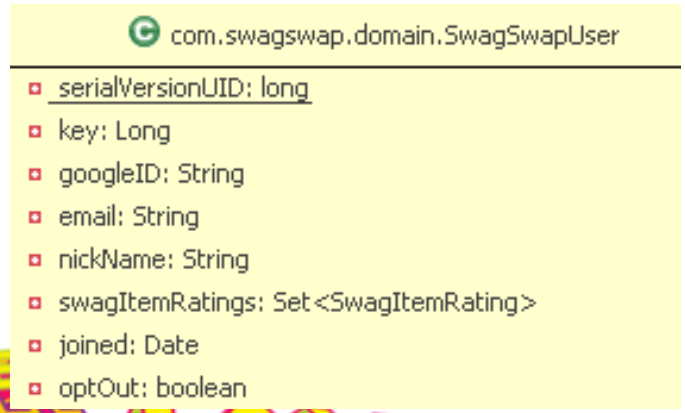
com.google.appengine.api.users.User

- Otherwise known as the Appengine User
- Always available to us in all tiers of the app through the com.google.appengine.api.users.UserService :

Method Summary	
java.lang.String	**createLoginURL**(java.lang.String destinationURL) Returns an URL that can be used to display a login page to the user.
java.lang.String	**createLoginURL**(java.lang.String destinationURL, java.lang.String authDomain) Returns an URL that can be used to display a login page to the user.
java.lang.String	**createLogoutURL**(java.lang.String destinationURL) Returns an URL that can be used to log the current user out of this app.
java.lang.String	**createLogoutURL**(java.lang.String destinationURL, java.lang.String authDomain) Returns an URL that can be used to log the current user out of this app.
User	**getCurrentUser**() If the user is logged in, this method will return a User that contains information about them.
boolean	**isUserAdmin**() Returns true if the user making this request is an admin for this application, false otherwise.
boolean	**isUserLoggedIn**() Returns true if there is a user logged in, false otherwise.

SwagSwapUser

- When a user logs in we get the Appengine User from Google Accounts, but we need our own.
- SwagSwapUser doesn't wrap Appengine User.
- we use the Appengine User to fetch SwagSwapUser as needed by googleID
- We were using email as the key but realized that the GWT app was exposing that to the client



SwagSwapUserService

- Wraps the Appengine UserService
- Exposes CRUD operations on SwagSwapUser
- Uses the MailService to send users a welcome email
- BlackLists users (so watch out)

Spring MVC Implementation

Click around in the Spring MVC implementation

Spring MVC Implementation

- The reference implementation
- Was super quick to develop
- Is decidedly Web 1.0 (for simplicity and stability)
- Uses Annotated Web MVC Controllers (POJOS)
- RESTful features of Spring MVC 3
- The views are JSPs with JSTL, Spring Tags, and custom taglibs
- Some Spring Controllers are used in all three client-side implementations

Annotated Web MVC Controllers

- Next-generation multi-action controllers
- Barely any configuration

```
<context:component-scan base-
  package="com.swagswap.web" />
```

- `@Controller` stereotype indicates that the methods should be scanned for request mappings
- `@RequestMapping` is specified at method level
- This binds a method to an HTTP path in the Spring **`DispatchServlet`**

Annotated Controller Example

```
@Controller
public class ItemController {
@Autowired
private ItemService itemService;
…
@RequestMapping(value = "/add", method =
  RequestMethod.GET)
public String addHandler(Model model) {
   model.addAttribute("swagItem", new
      SwagItem());
   return "addEditSwagItem";
}
…
```


RESTful Spring MVC 3 – URI Templates

- A URI template is a URI-like string, containing one or more variable names
- When these variables are substituted for values, the template becomes a URI:

```
@RequestMapping(value = "/edit/{key}",
 method = RequestMethod.GET)
public String editHandler(@PathVariable
  ("key") Long key, Model model) {
SwagItem swagItem = itemService.get(key,
  true);
…
```


Why Is This Great?

- Bookmarkable URLS
  - We sent emails with links to swagItems
- Much less code
- Handler methods have flexible signatures:
  - Just declare what you need:
    - Session, HttpRequest/Response, Model, Input/ OutputStream, RequestParam, Errors, and more
- Let's look at some examples

The JSPs

- Use Core JSTL and Spring taglibs
- Use Custom tags like:

```
<google-auth:isAllowed
  swagItemOwnerGoogleID="
  ${currentObject.ownerGoogleID}"
>
```

- Use the display-tag for showing sortable pageable tables

Custom Taglibs

- To keep Java out of the JSPs
- Spring service integration through extending **`AbstractSpringContextLookupTag`**
- Tags are:
  - **`IsAdminTag`**
  - **`IsAllowedTag`**
  - **`IsLoggedInTag`**
  - **`IsNotLoggedInTag`**
  - **`LoginLogoutTag`**
  - **`ShowRatingStarsTag`**

Getting SwagImages

- There are four ways for users to add SwagImages
  - FileUpload
  - URL
  - I'm Feeling Lucky
  - Email attachment
- **`ItemServiceImpl.populateSwagImage()`** looks in the **`SwagItem`** for **`imageBytes`** or an **`imageURL`** and uses what it finds

Getting Image From a URL

- ItemServiceImpl.getImageDataFromURL()
  - Uses the Appengine URL Fetch Java API
  - This costs us in outgoing and incoming bandwidth

```
URL url = new URL(imageURL);
bis = new BufferedInputStream
   (url.openStream());
```

- Retrieves url and turns it into a byte[] for storing in the SwagImage

FileUpload Gotcha

- The fileupload uses commons fileupload (via the Spring **`MultipartResolver)`**
- The problem is that Appengine doesn't allow you to write files to the server
- I found an in-memory version that did the trick
- It just holds a map of them in memory and exposes it to the spring context
- It is attached by
  - registering a custom editor for byte[] in the ItemController
  - And registering the **`MultipartResolver`** in the dispatcher-servlet.xml

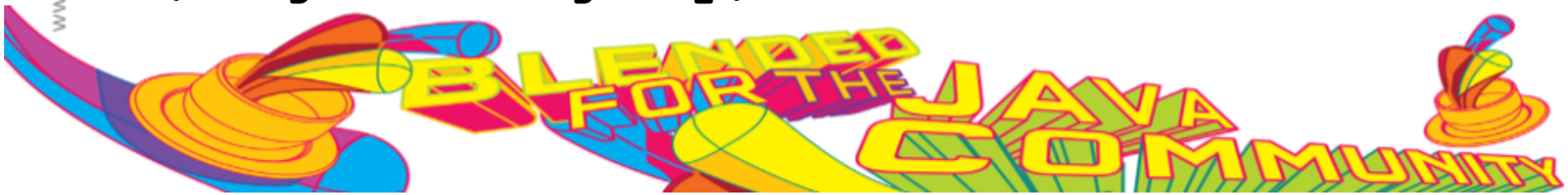
Streaming Images From the DB

- To stream SwagImages from the database we make them available via a RESTful URL

```
@RequestMapping(value = "/showImage/{key}",
  method = RequestMethod.GET)
...
outputStream.write(swagImageBytes, 0,
  swagImageBytes.length);
```

- Then in the client code, just treat the URL as an image tag:

```
<img border="0" src="<c:url value='/
  springmvc/showImage/$
  {swagItem.imageKey}'/>" />
```

What To Do If There's No Image

- If a SwagItem is added with no image
  - We leave the image null in the DB
  - We serve a default image (no_photo.jpg) in **`ImageController.streamContent()`**
- This uses **`ImageService.getDefaultImageBytes()`**
  - GOTCHA! - Painful to write because javax.ImageIO is blacklisted ☹

I’m Feeling Lucky Images

- The Spring and JSF versions use the Google AJAX Search API
- These are based on the raw search example: http://code.google.com/apis/ajax/playground/?exp=search#raw_search
- Raw search leaves the results rendering up to you
- The GWT version uses a Yahoo search (since I was able to borrow that quickly from a SmartGWT example)

Resizing Images

- Uses the Appengine Images Java API

```
ImagesService imagesService =
  ImagesServiceFactory.getImagesService();
Image oldImage = ImagesServiceFactory.makeImage
  (imageBytes);
Transform resize =
  ImagesServiceFactory.makeResize(resizedWidth,
resizedHeight);
Image newImage = imagesService.applyTransform
  (resize, oldImage,
ImagesService.OutputEncoding.valueOf("JPEG"));
```


Image Thumbnails

- Also uses the Appengine Images Java API
- Simply passes in a smaller size to the resize method

Exception Handling

- Uncaught exceptions from the webapp are configured in web.xml to show uncaughtException.jsp

```
<error-page>
  <exception- type>java.lang.Exception
  </exception-type>
  <location>/WEB-INF/jsp/
  uncaughtException.jsp
  </location>
</error-page>
```

- The JSP shows the exception and links to the Google Code report an issue page

SwagSwap Admins

- An admin on an Appengine app is registered (and has confirmed) under developers in the Appengine console

Google Account	Status	Remove Access
SamBrodkin@gmail.com	Active	Remove The bi
stevenson.scott@gmail.com	Active	Remove
swagswap.devoxx2009@gmail.com	Active	Remove

Invite a Developer to Collaborate on this Application:

Enter a complete email address

Invite

Admin Protection

- Is only available to SwagSwap admins
- Protected resource in web.xml

```
<security-constraint>
   <web-resource-collection>
      <url-pattern>/springmvc/admin/*</url-
     pattern>
   </web-resource-collection>
   <auth-constraint>
      <role-name>admin</role-name>
   </auth-constraint>
</security-constraint>
```


The Admin Page

- Populate Test Swag Items
  - Adds n items with a certain tag
  - Can timeout (30 second request limit)
  - How could we remedy this?
- Delete Test Items
  - Deletes items with the certain tag
- Blacklist user

GAE Cost Button

- Appengine sends out Live HTTP headers with cost information for each request (if you're logged in as admin)
- These are special non-standard HTTP headers
- There's a firefox plugin to read these but we just use javascript
- If you click on the "Show GAE Usage for this page" link in the deployed environment
  - We re-request the page and all it's unique images and total the cost:

```
response.getHeader('X-AppEngine-Estimated-CPM-US-Dollars')
```


JSF 2.0 Implementation

What's this demo about

SwagSwap JSF 2.0 Aims

- Provide a richer user experience than MVC implementation
- Use new features of JSF 2.0
- Demonstrate limitations of JSF 2.0 on App Engine
- Share common service layer with other implementations (injection of Spring beans)
- Use templating and composite components for consistent look and feel

JSF 2.0 Highlights

- No more JSF "Config Hell". Uses implicit navigation and annotations for Managed Beans and Custom Components.
- Easy composite component creation. No Java code or configuration required
- GET Support and bookmarkable URLs
- Built in Ajax support with <f:ajax> tag

JSF 2.0 Highlights (cont.)

- More efficient state-saving. Only state deltas are stored
- New Managed Bean scopes, View scope, Flash scope and custom scopes
- Templating with built-in Facelets support

JSF 2.0 GAE Restrictions

- Doesn't work "out of the box"
- Threading must be disabled in web.xml
- Requires hacked **`WebConfiguration`** class for threatened JNDI lookups
- Problems with Server-side state saving. SwagSwap uses Client state saving.

JSF 2.0 GAE Restrictions (cont.)

- Problems with Session and Application scoped beans. Restricted to Request Scope and View Scope.
- All state must be serializable in View scope.
- Component Binding not possible in View scope as base components not serializable.
- RichFaces and IceFaces cannot be used due to non-whitelisted classes.

SwagSwap Managed Beans

- **`SwagBean`** View scoped bean managing List View
- **`SwagEditBean`** View scoped bean managing Editing, Viewing and Adding of Swag Items
- **`SwagStatsBean`** Request scoped bean managing Stats view
- **`ActionBean`** Request scoped bean managing interaction with Service Layer

SwagSwap File Upload

- No File Upload component. Had to write custom renderer
- Uses Multi-part Servlet Filter **`SwagServletFilter`**
- Commons Fileupload uses non-whitelisted classes. Had to create extension for Memory Only File Item.

SwagSwap Composite Components

- **`<swag:stars>`** For entering and displaying ratings
- **`<swag:panel>`** For layout
- **`<swag:swagTable>`** List of Swag Items
- **`<swag:tableScroller>`** Ajax table scroller for data tables
- **`<swag:progressBar>`** For displaying percentages

JSF 2.0 Summary

- Work involved to get it up and running
- No third-party component support yet so be prepared to write extra components
- Operate within state saving and scoping restrictions
- It works!

(Smart)GWT Implementation

Click around in the GWT implementation

Google Web Toolkit (1)

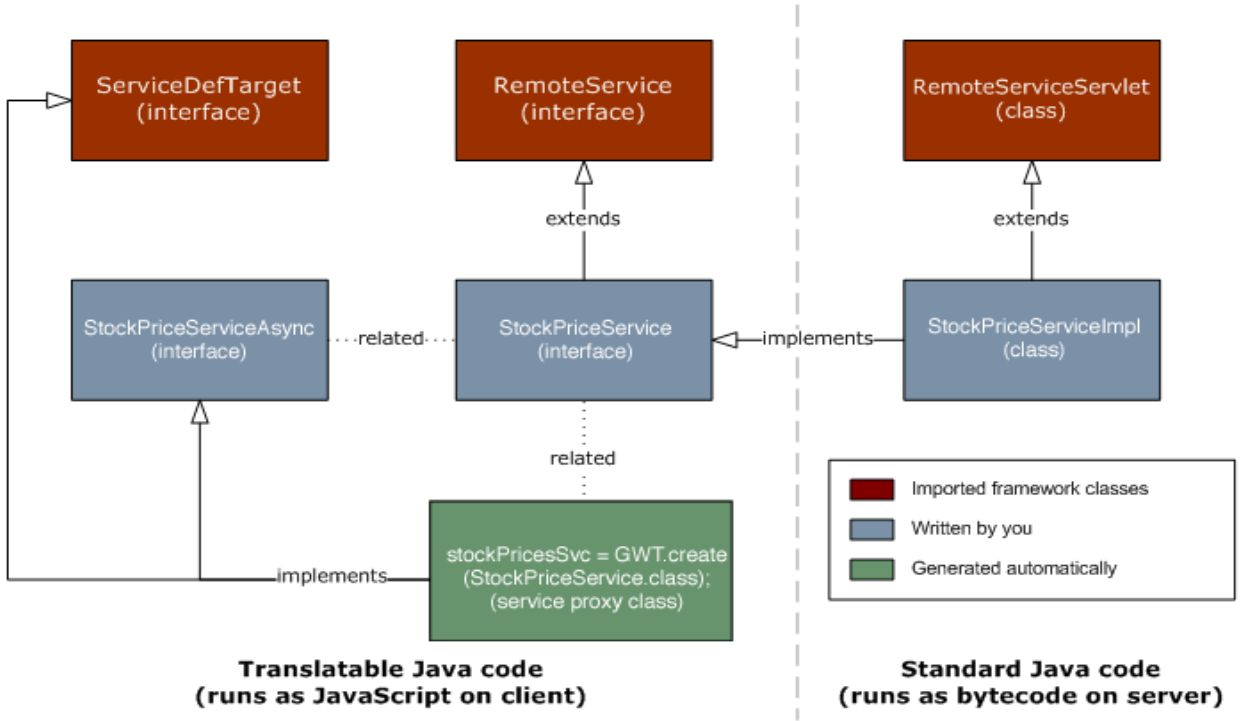
- Allows you to write an AJAX app entirely in (GWT) Java
- Cross-compiles into optimized JavaScript that works in all browsers
- Has a development mode where you can refresh and see changes immediately
- It also allows you to step through the Java abstraction of your AJAX code in a debugger

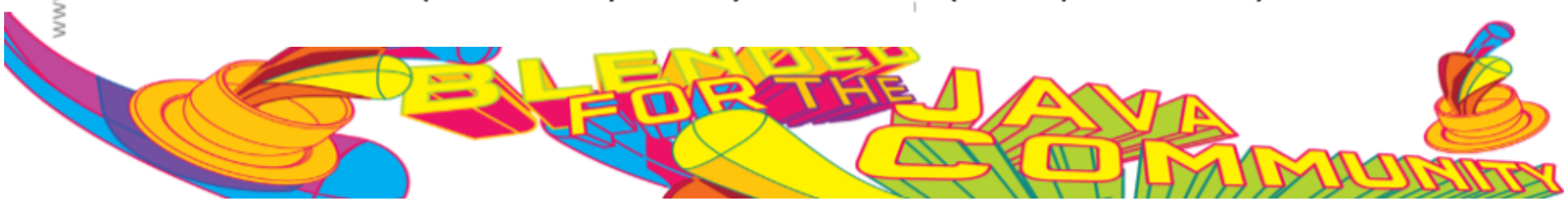
Google Web Toolkit (2)

- Your backend stays in pure server-side Java
- Communicate with the backend through GWT-RPC
- Can run on top of other Javascript libraries
- Swagswap uses SmartGWT which is built on top of the SmartClient library
- SmartGWT provides flashy components with a way to tie them in to your backend

Client vs Server Code

GWT RPC connects the client side GWT classes and the server-side Java





Appengine GWT gotcha

- Domain objects need to be serializable by GWT to pass over GWT-RPC
- Our domain object are JDO enhanced
- Is this a problem?
- Generally no problem, however we're using some Appengine specific types (Blob and Text). They don't GWT RPC serialize
- JDO backed collections are also a problem
- A possible workaround was to use Gilead but it didn't work
- Solution: pragmatic but a violation of the DRY principle: use DTOs

GWT and Spring

- The simplest solution I found for injecting our Spring services into my GWT server side classes was P.G Taboada's AutoinjectingRemoteServiceServlet
- Extending it allows your GWT service implementations to use Spring annotations for service injection from the spring context

The Services: LoginService

- Wraps what we need from SwagSwapUserService
- Returns a LoginInfo objects which wrap SwagswapUser objects
- GWT does an async call onModuleLoad which fetches and holds LoginInfo for subsequent use

The Services: ItemServiceGWTWrapper

- Wraps ItemService
- Translates between domain objects and GWTDTOs
- Also takes care of cloning collections to make sure none of them are JDO backed (or they can't be serialized by GWT RPC)
- I had to make sure to exclude domain classes with non-GWT RPC serializable attributes:
  in Domain.gwt.xml:

```
<source path="domain"
excludes="SwagImage.java,SwagItem.java,
SwagItemComment.java,SwagStats.java" />
```


Speaking of Violating the DRY Principle…

- SmartGWT grids (databound components) only operate on SmartGWT Records
- Records represent javascript objects in the underlying SmartClient library
- So I had to copy the SwagSwapDTOs into SmartGWT records and back.
- A Double DRY principle violation. Ugh.

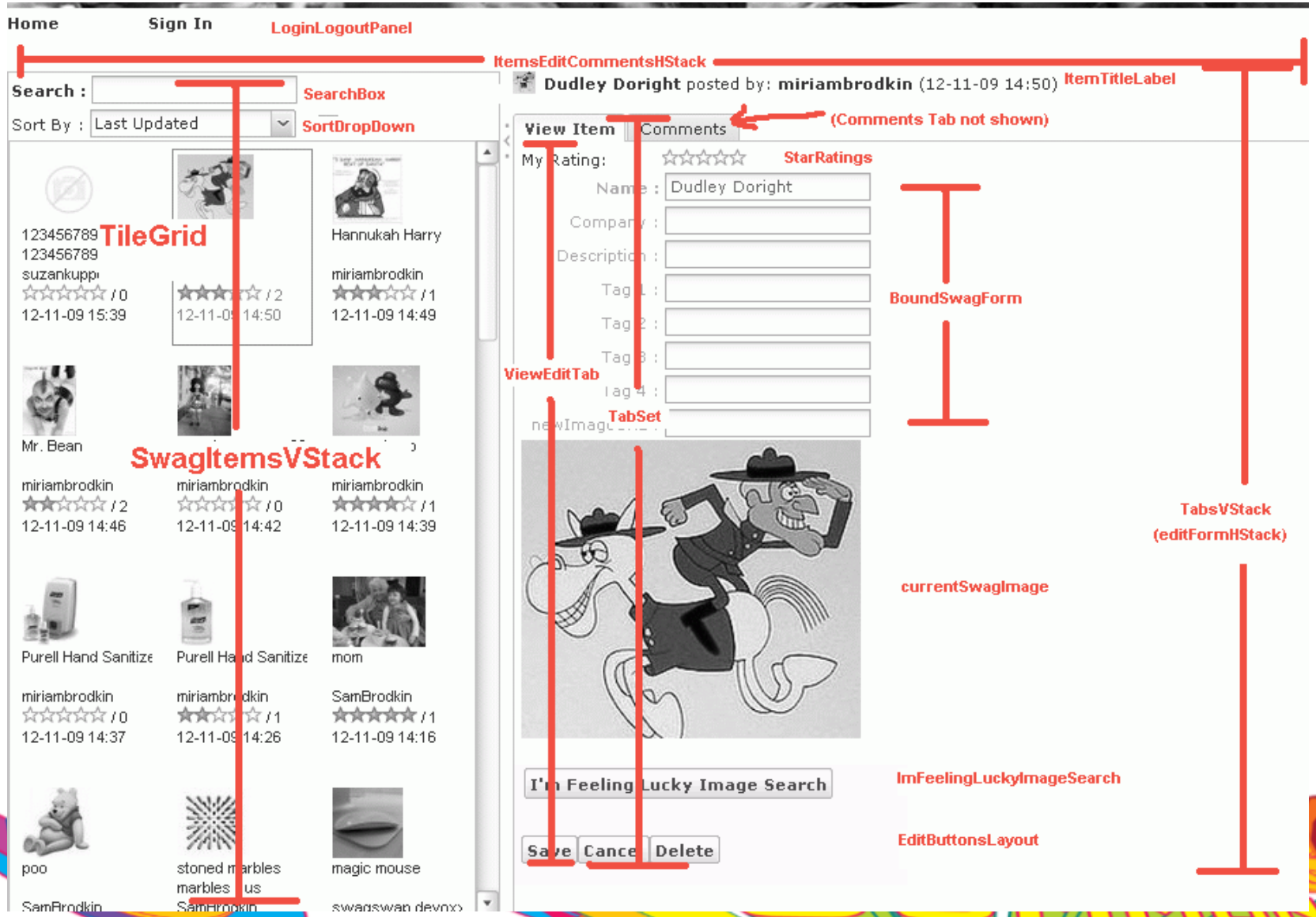
SmartGWTRPCDatasource

- Client helper that makes remote calls to ItemServiceGWTWrapper
- It conforms to the requirements of a SmartGWT DataboundComponents
- Translates between GWTDTOs and SmartGWT Records (objects behind a DataboundComponent)
- Has even more “repeating yourself” since it has to add fields by name which are bound to and displayed in DataboundForms
- Takes care of tricking the browser cache (changed SwagImages weren’t reloading since the URL never changes)

The GUI - SwagSwapGWT

- A monolithic GWT class which holds the entire implementation
- No reusable components so no need to refactor it (yet)
- Checks to see if the user is logged in and if so populates and holds a LoginInfo object
- Builds the GUI and loads data into the SmartGWT TileGrid
- Contains all the handlers for user interaction
- Uses the SmartGWTRPCDatasource and the GWT RPC services directly to communicate with the backend

The GUI – Visual Overview



The GUI - LoginLogout

- GWT can't access authentication information from the server synchronously (it's all async calls)
- So we need another way to get at the info
- **`SwagSwapGWT`** starts off by calling **`loginService.login()`** (async call to server)
  - returns a **`LoginInfo`** object that is stored and used subsequently
- **`buildGUI()`** is called when login() returns which starts by calling **`createLoginLogoutPanel()`**
  - Returns **`loginPanel`** or **`logoutPanel`** depending on **`loginInfo.isLoggedIn()`**

ItemsTileGrid – More Repeating Myself But Flashy!

- The SmartGWT TileGrid is the the DataBoundComponent that makes the SwagItems fly around
- It is bound the the SmartGWTRPCDataSource
- It has to have fields set which correspond to it's DataSource fields
- TileGrid has methods for
  - CRUD operations on Records (via the DataSource)
  - (de)selecting record
  - Searching and sorting data (locally)

ItemsTileGrid – onRecordClick()

- Copy values from the clicked TileRecord to a SwagItemDTO
- Check LoginInfo to see if they own the item
- Update the item label (over the edit / comment tabs)
- Select Edit tab
- Prepare the StarClickHandlers (set currentItemKey)
- Show View or Edit

The Search Box

- Uses a SmartGWT **`DynamicForm`**
  - Bound to the same DataSource as TileGrid
  - Contains SmartGWT FormItems

```
TextItem nameItem =
  new TextItem("name", "Search");
 // case insensitive
nameItem.setOperator(OperatorId.ICONTAINS);
```

- Uses the DynamicForm's Criteria to populate the TileGrid according to search input

The Sort Box (More Repeating)

- Creates a dropdown with items that match the itemsTileGrid items
- Has a checkbox for sortDirection (asc/desc)
- Calls:

```
itemsTileGrid.sortByProperty(
        sortValue,
        sortDirection
);
```


Add Swag

- Clicking this label calls the addSwagLabel ClickHandler
- This sets the currentSwagImage to no_photo.jpg
- Deselect all items in the ItemsTileGrid
- Removes the comments tab
- Focuses on name field
- Calls `boundSwagForm.editNewRecord();`
- See the SaveButton ClickHandler to see how the item is saved

Edit Item Form

- A SmartGWT BoundForm
- A SwagImage
- A Yahoo search based on entered item name
- Save, Cancel, and Delete buttons
- Calls operations on the boundForm
  - saveData()
  - clearValues()
- Delete button has a confirm popup
  - Eventually calls `itemsTileGrid.removeData()`
- Missing image upload functionality

Image Caching Gotcha

- When a SwagImage is updated the image URL doesn't change
- To trick the browser cache (and show the updated image in the TileGrid) I had to do a hack
- Append a random QueryString to image URLS on update
- Look at the **`SmartGWTRPCDataSource UpdateOrFetchCallback's call to appendRandomQueryString()`**

Star Ratings

- Star click are handled by the StarClickHandlers
- These are pre-populated with the currentItemKey
- If the user is not logged in, clicking the stars sends them to the login page
- Otherwise call
  `loginService.addOrUpdateRating()`
- `onSuccess()`:
  - Update LoginInfo with new rating
  - `updateUserRatingStars()`
  - Refresh the TileRecord (so that the new average rating shows)

Comments Tab

- Has a SmartGWT RichTextEditor
- A ListGrid displaying the comments
- Notice the label on top of the tabs
- When an TileRecord is clicked, the item is re-fetched to show up to date comment (refreshComments())
- Only shows the Add Comments RichTextEditor and Save button if they're logged in

SmartGWT Gotcha

- Smart GWT requires more than 1000 static resources
- The Appengine limit per app is 3000
- This put us over the limit
- The solution was to
  - zip the resource file (1000+ files → 1 file)
  - Create a servlet which pulls resources out of the zip file and caches them as needed
  - Exclude SmartGWT files from static resources in appengine-web.xml
- See **`SCServlet`**

The Animated Loading GIF

- Created as a in the SwagSwapGWT.html
- The status message is updated in various places in SwagSwapGWT

```
DOM.setInnerHTML(RootPanel.get
  ("loadingMsg").getElement(),"Fetching
  Swag Items");
```

- When the app finishes loading the is removed

```
DOM.setInnerHTML(RootPanel.get
  ("loading").getElement(),"");
```


Caching

What's this demo about

Caching

- Google Memcache
- Why use the cache?
  - Performance
  - Costs. A cache request is less expensive
- Implements **`javax.cache`** JCache standard
- Provides Map-like interfaces to store and retrieve data
- Configurable item expiration

MemCache on SwagSwap

- SwagItems, Images and Thumbails are cached
- All SwagItems are pre-loaded into cache. Images are lazy loaded.
- **`SwagCacheManager`** creates the Cache
- **`ImageCacheManager`** and **`ItemCacheManager`** responsible for orchestration
- Cache managers are injected implementations of DAOs.

Mail

Show places in the code that trigger getting a mail

Outgoing Mail

- Easy with Appengine!
- Appengine free quota is 2000 outgoing mails a day

```
Properties props = new Properties();
Session session =
  Session.getDefaultInstance(props, null);
Message msg = new MimeMessage(session);
msg.setFrom(new InternetAddress
  (from@gmail.com));
…
Transport.send(msg);
```


Task Queues

- Applications can perform work outside of a user request but initiated by a user request
- If an app needs to execute some background work, it may use the Task Queue API to organize that work into small, discrete units, called Tasks
- The app then inserts these Tasks into one or more Queues
- App Engine automatically detects new Tasks and executes them when system resources permit

Outgoing Mail With Task Queues (1)

- To prevent having to wait for sending mails, we send some mails with Task Queues:

```
//only can pass String or byte[] params
QueueFactory.getDefaultQueue().add
 (TaskOptions.Builder.url("/springmvc/
 admin/sendMail”)
                .param("swagItemKey",
                    swagItemKey.toString())
                .param("subject", subject)
                .param("msgBody", msgBody)
                );
```


Outgoing Mail With Task Queues (2)

- The Appengine task manager calls back a URL in our app
- Passes along specified request parameters
- The **`AdminController`** exposes a URL **`@RequestMapping`** to handle the call from the Task Manager
- It uses the swagItemKey to lookup the owner and emails them
- See **`AdminController.sendMailByID()`**
- **`Don't forget response.setStatus (HttpServletResponse.SC_OK);`**

Task Queues and the Dev App Server

- On the development server, task queues are not processed automatically
- Instead, task queues accrue tasks which you can examine and execute from the developer console
- Go to

`http://localhost:8080/_ah/admin/taskqueue`

- select the queue by clicking on its name
- Then select the tasks to execute.

Task Queues Demo

Show Executing Task on the Dev App Server

Incoming Mail

- Was just added in Appengine 1.2.6
- Allows your application to receive mails at

***string*@*appid*.appspotmail.com**

- Enable incoming mail in appengine-web.xml:

```
<inbound-services>
  <service>mail</service>
</inbound-services>
```

- **Email messages are sent to your app as HTTP POST requests using the following URL:**

/_ah/mail/<address>

Incoming Mail in SwagSwap

- When a user signs up, they get instruction on how to send an email SwagItem to SwagSwap
- Send mail to **`add@swagswap.appspotmail.com`**
- Put the item name in the subject
- Attach an optional image
- I wanted to use the mail body for the description but was getting `java.lang.OutOfMemoryError` when sending from the Mac
- See **`IncomingMailController`**
- You can simulate incoming emails in the Dev App Server (but not with attachment)

Storing Data

Storing Data

- The Appengine datastore uses Google BigTable
- It is not relational, however Appengine provides a JDO and JPA layer on top
- The App Engine datastore stores and performs queries over data objects, known as *entities*
- An entity has one or more *properties*, named values of one of several supported data types
- A property can be a reference to another entity
- Each entity also has a *key* that uniquely identifies the entity

JDO (Java Data Objects)

- I chose JDO since that's what all the examples use
- JDO uses annotations on POJO domain objects to describe
  - How instances of the class are stored in the datastore as entities
  - How entities are recreated as instances when retrieved from the datastore:

```
@NotPersistent
private String imageURL;
@Persistent
private String ownerGoogleID;
```


JDO PersistenceManager

- How you interact with the datastore
- Configured in Spring and injected into DAOs
  - DAOs extend Springs JdoDaoSupport
  - This provides a JdoTemplate to DAOs
  - Takes care of re-using, and closing PersistanceManagers
- JDOTemplate
  - Simplifies JDO data access code
  - Converts JDOExceptions into Spring DataAccessExceptions

JDO Relationships

- Owned unidirectional one-to-one relationships
  - Just declare an attribute of another domain type
- One-to-many relationships
  - Declare an attribute of type Collection
- The rest you have to model yourself by holding a foreign key or a Collection of keys
- If a parent is deleted, it's children are too (cascading deletes)

Entity Keys

- Every entity has a key that is unique over all entities in App Engine
- A complete key includes several pieces of information, including:
  - The application ID
  - The kind
  - An entity ID
  - information about entity groups
- Identify the primary key field using the @PrimaryKey annotation

Entity Key Types

- There are different possible entity key types
- We use the Appengine Key encoded as String
- This is portable (using Appengine Key type is not)
- The key value includes
  - The key of the entity group parent (if any)
  - And a system-generated section
- Looks like this for a parent:

aghzd2Fnc3dhcHIPCxIIU3dhZ0l0ZW0YtH0M

- And this for it's child:

aghzd2Fnc3dhcHIeCxIIU3dhZ0l0ZW0YtH0MCxIJU
3dhZ0ltYWdlGAEM

Annotating Keys

- Parent entities have Long keys with the following JDO annotations:

```
@PrimaryKey
@Persistent(valueStrategy =
  IdGeneratorStrategy.IDENTITY) //auto populated
private Long key;
```

- Child keys are Strings (so that they can include parent's key)
- They use an Appengine annotation extensions:

```
@PrimaryKey
    @Persistent(valueStrategy =
  IdGeneratorStrategy.IDENTITY)
    @Extension(vendorName="datanucleus",
  key="gae.encoded-pk", value="true")
    private String encodedKey;
```


Persisting and Getting Objects

- For insertion of a new object:

```
getPersistenceManager().makePersistent(
 swagItem);
```

- For update:

```
//Fetch swagItem
SwagItem swagItem = getPersistenceManager
  ().getObjectById(
   SwagItem.class, id);
swagItem.setLastUpdted(new Date());
```

- That's it! Fetched item is attached so changes made to it are persisted when transaction is committed by Spring

Deleting Objects

```
PersistenceManager pm = getPersistenceManager
  ();
SwagItem swagItem = pm.getObjectById
  (SwagItem.class, id);
pm.deletePersistent(swagItem);
```


JDO Queries (JDOQL)

```
protected List<SwagItem> findByName(String
  searchString) {
  Query query =
    getPersistenceManager().newQuery(
    "select from " + SwagItem.class.getName() +
    " where name==p1 parameters String p1");
   return (List<SwagItem>)
     query.execute(searchString);
}
```


JDO Query GOTCHA

- There is currently no way to do a case-insensitive query on the Appengine datastore
- Possible workarounds:
  - Store all data in lowercase
  - Use Compass (Lucene)
  - Filter client-side
  - Use Scott's pragmatic solution (See **`ItemDaoImpl.search()`**)

Transactions

- JDO requires an active transaction to modify a persistent object
- We're using Spring declarative transaction management

```
<bean id="txManager"
   class="org.springframework.orm.jdo.JdoTransactionManager
   ">
    <property name="persistenceManagerFactory"
      ref="persistenceManagerFactory" />
</bean>

<tx:annotation-driven transaction-manager="txManager" />
```


Declaring Transactions

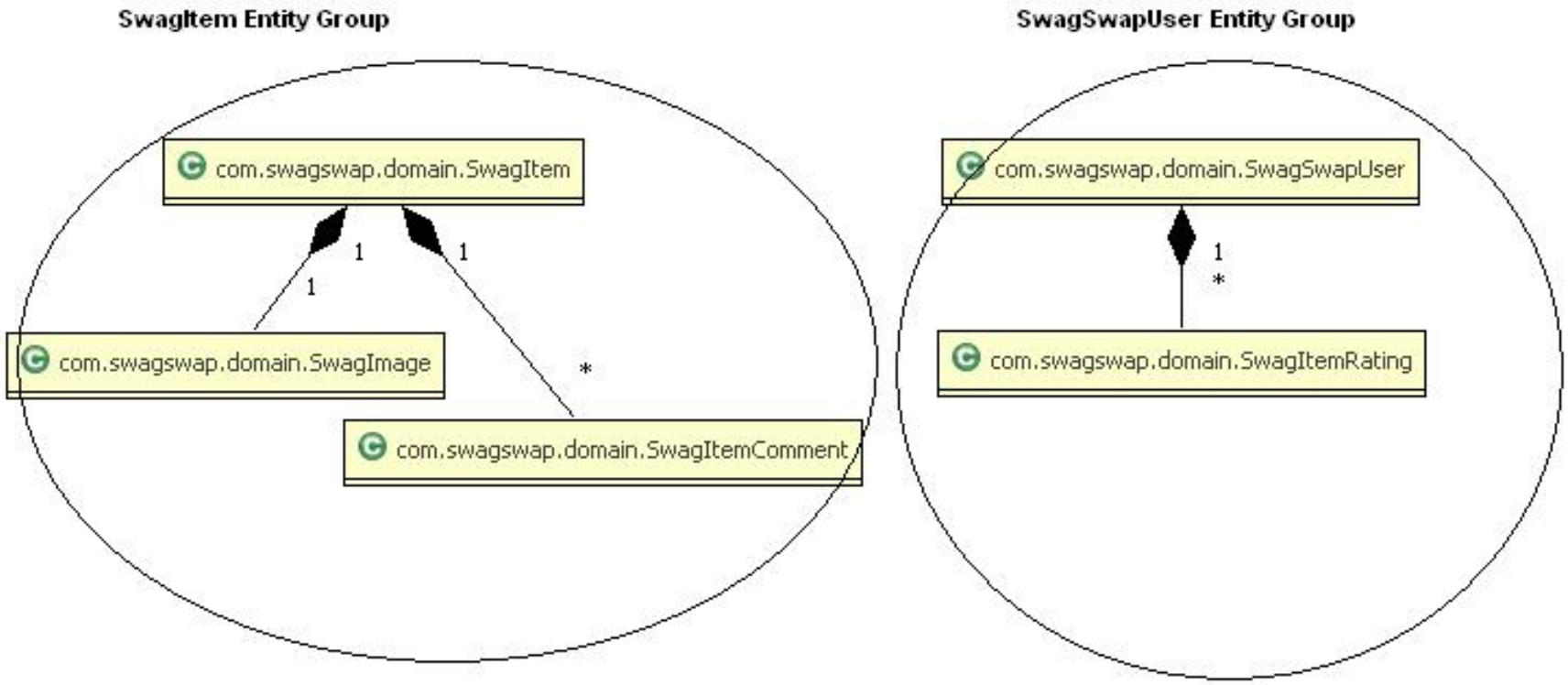
- Transactions can be declared in any tier of the application
- We did it in the service layer
- ItemServiceImpl:

```
@Transactional(readOnly = false, propagation =
  Propagation.SUPPORTS)
public SwagItem save(SwagItem swagItem) {…}
```


Transactions and Entity Groups

- Entity groups are defined by a hierarchy of relationships between entities
- To create an entity in a group, you declare that the entity is a *child* of another entity already in the group
- A child's key contains it's parent's key
- An entity created without a parent is a *root entity*
- GOTCHA - You can only operate on one entity group within a single transaction !!!

Entity Groups in SwagSwap (1)



Entity Groups in SwagSwap (2)

- You can't update a SwagItem and a SwagSwapUser in the same transaction
- So this is illegal :

```
@Transactional(readOnly = false, propagation =
  Propagation.SUPPORTS)
public SwagItem save(SwagItem swagItem) {
  itemDao.insert(swagItem);
  SwagSwapUser swagSwapUser =
   findCurrentUserByEmail();
  userDao.insert(swagSwapUser);
}
```

- You could do it in two separate calls from the web tier


BLENDED
FOR THE
JAVA COMMUNITY
WEB
AJAX
LOUD
SECURITY
BPM
RIA
JAVA
SCRIPTING
DEVOX
AJAX
AGILE

Tooling

Tooling

- Use Google Code
  - For SVN Repo
  - Issue Tracking (With SVN integration)
  - Wiki
  - Release announcements
- Eclipse
  - Prefer ANT over GAE and GWT plugins
- Google Wave for brainstorming
- Google Talk for nearshoring to Belgium and outsourcing to America (Mom was a tester)

Issues Integration with SVN

- Create an issue in Google Code project for every feature and bug
- Commit enhancements or fixes with the following in the SVN commit comment:
  - Fixes issue NNN <issue title>
  - Update issue NNN <update description>
- This closes or updates an issue with the committed diffs

Releases

- Label issues with upcoming release number
- On release
  - Tag the project in SVN with release number like 0dot6
  - Do a query on the issues for all Fixed issues with label 0dot6
  - Use the link to that query as the release notes
  - Announce the release on the project home page
  - Use issue bulk update to add a label to all open issues for the next release like 0dot7

Deployment

- Use appengine-web.xml to specify the version to deploy
- Version can be any String but cannot contain “.”’s
- Deploy the version, test, and make it the default (live) version
- Sometimes cleaning the DB is necessary
- Can have multiple versions deployed at once so if the DB model changes, old incompatible versions should be deleted

Unit Testing

- The Appengine dev server comes with a TestEnvironment

```
class TestEnvironment implements ApiProxy.Environment
  {…}
```

- Base LocalDataStoreTestCase uses this
  - Provides a idempotent way to test DAOs locally:

```
LocalDatastoreService.NO_STORAGE_PROPERTY,
  Boolean.TRUE.toString()
```

- Use `countEntities()` in assertions

Create your very own GAE Application

Thanks for listening!

Sam Brodkin
sam@brodkin.com

Scott Stevenson
scott@scottstevenson.com