

Contents

Item Viewer Service Technologies Overview	2
Item Viewer Service Modules	2
App	2
Core	3
Data Access Layer	4
Smarter Balanced Libraries	4
Dictionary	4
Iris	4
Third Party Libraries	5
Amazon Web Services Java SDK	5
Apache Commons	5
Logback Classic	5
Jackson Databind	5
Operating System Hardware Information	5
SLF4J	5
Spring	5
Configuration	6
Item Viewer Configuration	6
General	6
Logging	6
Tomcat Configuration	6
Item Viewer Service Manual Setup	7
Dependencies	7
Compile Time Dependencies	7
Run Time Dependencies	7
Configuration	7
Dictionary API	7
Aws Container Cluster Information (Optional)	8
Logging	8
Local System	8
Building and Running	8
Building	8
Running Locally	8
Running on an EC2 Instance	9
Installation	10
Tomcat Configuration	10
Configure nginx	10
Launch Application	11

Item Viewer Service Technologies Overview

Item Viewer Service Modules

The item viewer service provides an API to load a single content item and accommodations in a page. The item and accommodations are specified as URL parameters. The item viewer service is divided into three layers, the App, the Core, and the Data Access Layer or dal. Each layer is a Maven submodule that is part of the main item viewer service Maven application.

App

The App module has the web application frontend parts and application configuration. It contains the web application controllers, JavaScript, page templates, and application configuration files.

Spring

Spring is used for controllers and scheduled services. The item viewer service uses a mixture of xml and annotation configuration for Spring. Controllers use the `@RequestMapping` annotation to map the item and diagnostic API URLs. The item viewer service servlet is configured with xml and used to map the item page template directory and .jsp file extension.

The controller for loading items is mapped to `/item/itemID`. The item ID must match the regex `d+[-]d+`, that is one or more numbers, a dash, and one or more numbers in that order. Optional accommodations are specified with the `isaap` URL parameter and are semicolon delimited. Only the accommodation code should be specified. Accommodation type is not specified. For example, the URL to load item 200-12344 with the word list glossary and expandable passages would look like `http://viewer.smarterbalanced.org/item/200-12344?isaap=TDS_WL_Glossary;TDS_ExpandablePassage`

The Diagnostic API returns xml formatted diagnostic results per the diagnostic API requirements. The diagnostic API supports levels 0 through 5 as required in the requirements. The diagnostic level is specified as a URL parameter. For example the level 3 diagnostic API would be accessed with `http://viewer.smarterbalanced.org/status?level=3`. The diagnostic API is mapped to `/statusLocal`. If the Item Viewer is running in an AWS ECS cluster it can be configured to display the diagnostic status for each instance of the Item Viewer running in the cluster. The cluster status is mapped to `/status`.

The service that polls Amazon Web Services S3 for updated content packages and downloads them to the local file system is run as a Spring scheduled service. It is configured with annotations to run every 5 minutes after the previous run of the service has finished.

JavaScript

The item viewer service includes all of the JavaScript from Iris required to display items and accommodations.

Configuration

The App layer contains the logging and application configuration files.

Core

The Core module contains the application's business logic. It contains the diagnostic API, item request processing logic, and the tool that checks for updated content packages on Amazon S3.

Diagnostic API

The Diagnostic API is implemented using the requirements listed in the Smarter-App Web Diagnostic API documentation. It supports five levels of diagnostics; system, configuration, database read, database write, and external providers. The system diagnostic uses the Operating System Hardware Information (OSHI) library to gather information on memory usage and file system space. The configuration diagnostic checks for the existence of the application configuration file and configuration variables. The database read diagnostic makes sure the Iris content path variable is readable, and that it contains content items. The database write diagnostic makes sure that the Iris content directory is writeable, then performs a write and a delete in the content directory. The providers diagnostic checks the status of the word list handler, the black box, the item viewer service API, the Amazon S3 content bucket, and the content packages. It performs an HTTP get request to get the status of the word list handler, black box, and item viewer service API. It uses the Amazon AWS Java SDK to connect to Amazon S3 and get a list of content packages. The diagnostic API can be accessed at `/statusLocal?level=<1-5>`. Replace the brackets and number with a status level between 1 and 5 inclusive.

If the Item Viewer is running on an AWS ERS cluster it can be configured to display the diagnostic status of each instance of the Item Viewer in the cluster.

Item Request Translation

When the item viewer service receives a request for an item the request is translated into a JSON token that the Iris will accept. The item viewer service parses the item bank and key out of the URL as well as any accommodation codes. The Iris requires both the accommodation type and code for each accommodation. The item viewer service only requires codes. A reverse lookup is performed to

get the accommodation type for each accommodation code. Finally the item bank and key, and accommodation types and codes are serialized into a JSON token that Iris can parse.

Data Access Layer

The Data Access Layer contains the classes used to access configuration.

Smarter Balanced Libraries

Dictionary

The Smarter Balanced Dictionary is a runtime dependency of the Iris application, and therefore the item viewer service application. It provides an API that is used for the dictionary accommodation. The item viewer service requires that it is configured and running.

Iris

The Smarter Balanced Iris is used as a Maven WAR overlay to extend the scripts, styling and functionality of the Iris application into the item viewer service. The Iris application displays a window for users with a text box where they can enter a JSON token to load an item and accommodations. The item and accommodations are loaded in an iFrame embedded in the page with the text box. The iFrame with the items and accommodations is the front end part of the Iris that the item viewer service makes use of. It loads only the iFrame and selects which item and accommodations are loaded from the URL.

The item viewer service excludes some files from the Iris WAR overlay. It excludes the Iris web.xml file because it requires different servlet mappings. It excludes the JNA 3.0.9 jar because it causes a dependency conflict with the Operating System Hardware Information library which depends on JNA 4.2.2. Finally it excludes the IrisPages directory because it does not need the page templates it contains.

The item viewer service extends the Iris application by adding its own controllers for loading items and accommodations by URL, and the diagnostic API. In the backend it adds the diagnostic API logic, accommodation code to type lookup, and a service that fetches content packages from Amazon Web Services S3.

Third Party Libraries

Amazon Web Services Java SDK

The Amazon Web Services (AWS) Java SDK is used to connect to the AWS S3 service. It is used for authentication, and S3 file information and downloads.

Apache Commons

The Apache Commons IO library is used in the data access layer to take the file stream provided by the AWS Java SDK S3 client and write the file to disk. It is used to transfer and delete files when updating the content on the local file system.

Logback Classic

Logback classic is the logging framework used by the Iris.

Jackson Databind

Jackson databind is used to serialize the data from the API call to item viewer service into a token that can be sent to the Iris.

Operating System Hardware Information

The Operating system Hardware Information (OSHI) library is used by the system diagnostic to get information on total memory, memory usage, and file system size, usage and type.

SLF4J

SLF4J is the logging facade used by Iris. It can be bound to a number of different logging frameworks. In the case of Iris and the item viewer service the logback classic logging framework is used.

Spring

Spring is the web application framework used in Iris and other Smarter Balanced applications. The Item Viewer Service uses version 3.2.1 because that is the same version Iris uses.

Configuration

Item Viewer Configuration

General

The Item Viewer Service config file is located in `app/src/main/resources/settings-mysql.xml`. Most of the values are carried over from Iris. The following options must be configured in the `settings-mysql.xml` config file for the Item Viewer to function correctly. - `iris.ContentPath` must be set to the location of the content package on the local filesystem. - `iris.DictionaryUrl` must be set to the url of the dictionary API. - `AwsRegion` Set this to the AWS region the Item Viewer is running in if it is running on AWS. - `AwsClusterName` Set this to the AWS ECS cluster the Item Viewer Service is running in if it is running on AWS ECS.

Logging

The Item viewer service uses Logback Classic bound to SLF4J for logging. The Logging configuration file is `app/src/main/resources/logback.xml`. The config file included will log to stdout and `/home/tomcat7/itemviewerservice.log`. Details for configuring the log output can be found in the logback classic documentation.

Tomcat Configuration

The Item Viewer must be run in Apache Tomcat 7 or newer.

In order to run correctly the following Tomcat configuration needs to be set.

Set a 25 character alphanumeric numeric encryption key for Iris in `$TOMCAT_HOME/conf/context.xml` under the context element. The entry follows the form `<Parameter value="YOUR KEY ENCRYPTION KEY HERE" override="false" name="tds.iris.EncryptionKey"/>`.

The dictionary API call is made as a cross origin request. A CORS filter must be added to `$TOMCAT_HOME/conf/web.xml`.

```
<filter>
  <filter-name>CorsFilter</filter-name>
  <filter-class>org.apache.catalina.filters.CorsFilter</filter-class>
</filter>
<filter-mapping>
  <filter-name>CorsFilter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```

Item Viewer Service Manual Setup

To build and run the item viewer service on your local machine you will need the following.

Dependencies

Compile Time Dependencies

Compile time dependencies are built into the Maven POM file. Java 7 is required to build and run the Item Viewer Service.

The Item Viewer Service depends on the Iris package of TDS_Student.

Run Time Dependencies

- Apache Tomcat 7 or newer
- Smarter Balanced Dictionary API access
- Read access to the local file system

Configuration

The item viewer service configuration file is located at `app/src/main/resources/settings-mysql.xml`.

The `iris.ContentPath` variable in the `settings-mysql.xml` file needs to be set to the local directory where the content packages are going to be stored. If this directory does not exist, or the application can not access it, it will fail to launch.

Iris requires a 25 character alphanumeric numeric encryption key set as a parameter in `$TOMCAT_HOME/conf/context.xml` under the context element. The entry follows the form `<Parameter value="YOUR KEY ENCRYPTION KEY HERE" override="false" name="tds.iris.EncryptionKey"/>`.

Dictionary API

In order to use the dictionary you need to set the `iris.DictionaryUrl` value in the `settings-mysql.xml` config file for the Iris. The dictionary should be a running instance of the TDS_Dictionary application.

The dictionary API call is made as a cross origin request. A CORS filter needs to be added to the Tomcat `$TOMCAT_HOME/conf/web.xml` file.

```
<filter>
  <filter-name>CorsFilter</filter-name>
  <filter-class>org.apache.catalina.filters.CorsFilter</filter-class>
```

```
</filter>
<filter-mapping>
  <filter-name>CorsFilter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```

Tomcat has detailed documentation on setting up CORS filtering. Please refer to it if you want to set up a more detailed filter.

Aws Container Cluster Information (Optional)

The “AwsRegion” and “AwsClusterName” keys are used if the application is being run outside of an AWS ERS cluster. If you are running the Item Viewer locally you can ignore these. Please note that the /status url will not work if the Item Viewer is being run outside of an AWS ECS cluster. Use /statusLocal for local diagnostics.

Logging

The item viewer service uses SLF4J bound to Logback Classic for logging. The log settings are found in logback.xml. For basic logging to a file you will need to set the file location for the file appender. For a full reference on configuring the log output levels and locations please refer to the Logback Classic documentation.

Local System

The application requires read permissions to the Iris content directory specified in settings-mysql.xml.

Building and Running

Building

To build the item viewer run `mvn install` in the top level project directory. The compiled WAR file will be generated in `app/target/itemviewerservice.war`.

Running Locally

Deploy the WAR file to Apache Tomcat by placing the itemviewerservice.war file in your tomcat webapps directory. Restart Tomcat.

Running on an EC2 Instance

If you are running the Item Viewer in an EC2 instance you will need to configure it to

AWS Prerequisites

- Create a security group to allow access to certain ports:

Inbound

Type	Protocol	Port Range	Source
HTTP	TCP	80	0.0.0.0/0
SSH	TCP	22	0.0.0.0/0

Outbound

Type	Protocol	Port Range	Source
All Traffic	All	All	0.0.0.0/0

AWS Setup

Launch an Amazon Web Services instance with the following configurations:

1. Use AMI: Ubuntu Server 14.04 LTS (HVM), SSD Volume Type (ami-d732f0b7).
2. Select a suitable instance size.
3. Select **Next: Configure Instance Details**
4. Add the IAM role that grants S3 bucket access
5. Select **Review and Launch**.
6. Next to **Security Groups**, select **Edit Security Groups** and add the security group created in the **Prerequisites** section.
7. Launch your instance.

Dependency Installation

In the AWS instance launched, update packages: `apt-get update`

- Install openjdk-7: `apt-get install openjdk-7-jdk`
- Install tomcat7 and tomcat7-admin: `apt-get install tomcat7 tomcat7-admin`
- Install nginx for port forwarding: `apt-get install nginx`

Installation

After the AWS instance launches: - Update packages: `apt-get update`

- Install openjdk-7: `apt-get install openjdk-7-jdk`
- Install tomcat7 and tomcat7-admin: `apt-get install tomcat7 tomcat7-admin`
- Install nginx for port forwarding: `apt-get install nginx`

Tomcat Configuration

- Create a directory for tomcat give it permissions:

```
mkdir -p /home/tomcat7/content  
chown -R tomcat7:tomcat7 /home/tomcat7  
chown -R tomcat7:tomcat7 /usr/share/tomcat7
```
- Update the tomcat configuration files as mentioned above by adding the following to `/etc/tomcat7/web.xml`:

```
<filter>  
  <filter-name>CorsFilter</filter-name>  
  <filter-class>org.apache.catalina.filters.CorsFilter</filter-class>  
</filter>  
<filter-mapping>  
  <filter-name>CorsFilter</filter-name>  
  <url-pattern>/*</url-pattern>  
</filter-mapping>
```

Configure nginx

Replace `/etc/nginx/sites-available/default` with the following text (requires root permissions):

```
server {  
    listen 80;  
    location / {  
        proxy_pass http://localhost:8080;  
        proxy_http_version 1.1;  
        proxy_set_header Upgrade $http_upgrade;  
        proxy_set_header Connection keep-alive;  
        proxy_set_header Host $host;  
        proxy_cache_bypass $http_upgrade;  
    }  
}
```

Launch Application

Restart tomcat7 and nginx:

```
sudo service tomcat7 restart
```

```
sudo service nginx restart
```