

EVADe Web-app Technical Design Document

Neil Peirce

Trinity College Dublin

Version	Date	Contributors	Description
0.1	29/10/13	Neil Peirce	Initial Contribution
0.2	15/11/13	Neil Peirce	Updates due to refactoring work, and performance improvements
0.3	28/11/13	Neil Peirce	Updates to add Regex to queries and logging config interface

Table of Contents

Introduction	6
Purpose of Document	6
Overview of Document	6
Project Description.....	6
Project Scope	7
List of Figures	8
System Architecture.....	9
Data Design	9
Components Overview.....	11
Core Web Application	11
Platform	11
Technologies Used	11
Component Dependencies.....	11
Development Environment.....	11
Description	11
Web-app Architecture	12
Security and Authentication	13
File Structure	14
Configuration	15
Dependency Management.....	15
Testing.....	16
Installation and Deployment.....	16
Open Source Licenses	16
Online Documentation.....	17
Grails Models	18
Technologies Used	18
Description	18
UML Class Diagram	19
User	20
UserGroup.....	20
Role	20

UserRole.....	20
Source	20
Field.....	20
ApiAdapter	20
AdapterMapping	20
Rule	20
RuleFieldCondition	20
RuleFieldProcessing	20
RuleQuery	21
RuleBase.....	21
VisualisationEmbed.....	21
Visualisation	21
Grails Controllers	22
Technologies Used	22
LoginController	22
LogoutController	22
AdminController.....	22
AnalyticsController.....	22
ApiController.....	23
CaptureController	23
VisualisationController.....	23
Miscellaneous Controller Notes.....	23
Grails Views.....	24
Description.....	24
Technologies Used	24
Tag libraries.....	25
Grails Services	26
Technologies Used	26
Component Dependencies.....	26
ApiService.....	26
CompilerService	26
QueryService	26

RuleService.....	26
SourceService.....	27
StorageService	27
VisualisationService	27
CacheService	27
Non-grails Classes	28
Technologies Used	28
Component Dependencies.....	28
AjaxStream	29
ApiTarget.....	29
ControllerUtils	29
CryptoUtils	29
DataStore	29
FilteredStorageService	29
KMPDataStore.....	29
MongoDataStore.....	29
Utils	30
SaveFilterRuleCommand.....	30
CustomClassPool.java	30
CustomMySQLDialect.java	30
DummyHttpClient.java.....	30
DynamicClassLoader.java.....	30
FieldType.java	30
FilteredOpenSessionInViewInterceptor.java	30
HTTPClient.java	30
RuleCompiler.java	30
RuleConditionType.java	30
RuleProcessingType.java.....	30
RuleUtil.java	31
SourceClassCompiler.java	31
SourceOrigin.java	31
StorageException.java.....	31

AbstractData.java.....	31
Counter.java.....	31
User Interface Design.....	32
User Interface Overview	32
Workflow Examples	32
Super User.....	32
Normal User	39
Admin User	40
Visualisation Generation.....	41
Technologies Used	41
Supported Visualisation	42
Adding additional visualisations	43
Visualisation Embedding.....	43
Embedding Process.....	43
Example Visualisation Embed code for an embed.....	44
Known Issues.....	44
References	45
Internal Query Format	45
Constraint types	46
Browser SVG Support.....	47
Browser JSON Parsing Support	48
Full project file list.....	48

Introduction

Purpose of Document

This document details the architecture and components of the EVADE Web-app. The intended audience of this document are the partners of the Learnovate Centre.

Overview of Document

This document details the architecture of the EVADE web-app, the technologies used, and the component interactions within the system.

Project Description

Sector: Corporate

Challenge: As organisations increasingly seek to establish metrics around staff training and to demonstrate training impact, there will be an increasing demand for learning analytics tools that are easy to use by non-IT specialists such as L&D professionals.

To date efforts to mine learning data have been hampered by a lack of data mining tools that are easy for non-experts to use and by poor integration of data mining tools with e-Learning systems (Romero & Ventura, 2010).

One of the common challenges articulated by Learnovate's industry partners was the inflexibility of existing Business Intelligence (BI) suites. These were seen as being separate from the LMS or other applications in which learning was occurring. The existing solutions required users to leave the learning environment and go to a specific reporting platform where they could view preconfigured reports that were designed by experts. These solutions provide limited capabilities for the end user to configure their own learning analytics in order to query the system for information that is meaningful to them. The very nature of the reports generated by existing solutions is also a problem, they do not facilitate the surfacing of relevant information to different stakeholders as and when it is needed. The user has to go looking for the information rather than the information being brought to them.

Research Questions: Does using the Evade technology:(Note that the research questions map to the industry challenges)

- Provide L&D with configurable, flexible, agile and intuitive reporting capability?
- Improve process efficiencies in the L&D department?
- Reduce the time it takes for L&D to identify learner-related problems in the LMS?
- Reduce the time it takes for L&D to identify content-related problems in the LMS?
- Provide L&D with insight into course effectiveness and learner engagement?
- Enable L&D to provide timely, targeted learning interventions?
- Slow down the LMS?

(Feel free to provide additional/ alternative questions)

Project Scope

EVADE stands for End-to-end Visualisation and Analytics of Data for Elearning. As a system it enables the capture, analysis and visualisation of data related to learning through integration with LMSs, talent management systems, and other sources of learner evidence. The system provides an intuitive web-based interface to select how data is analysed and visualised, with the option to embed indicative visualisations within an LMS. The web-interface also provides access to powerful real-time Big Data analytics in order to understand and interpret the wealth of gathered learner data. EVADE can also act as a Learning Record Store (LRS) and can support the Tin Can/Experience API (<http://tincanapi.com/learning-record-store/>)

List of Figures

Figure 1. EVADE Overall Architecture	9
Figure 2 Grails web-app high-level architecture	12
Figure 3. Controller and Service Interaction in the web-app.....	13
Figure 4. Table of Open Source licenses used.....	17
Figure 5. UML Class Diagram of Domain Models.....	19
Figure 6. Web-app home page.....	32
Figure 7. Web-app login page	33
Figure 8. Data Capture API List page.....	33
Figure 9. Creating a new Data Capture API.....	34
Figure 10. Adding fields to a Data Capture API.....	34
Figure 11. Viewing example usage for a Data Capture API.....	35
Figure 12. Creating a filter on captured data.....	36
Figure 13. Viewing all filtered data sources.....	37
Figure 14. Adding a query to filtered data	37
Figure 15. Super User Visualisation Builder view	38
Figure 16. Editing an embed location	38
Figure 17. Normal user Visualisation Builder view	39
Figure 18. Example visualisation embedded within Moodle.....	40
Figure 19. Admin Settings	40
Figure 20. Mongo Data Store Admin	41
Figure 21. Visualisation loading symbols	42
Figure 22. Example visualisation errors	42
Figure 23. Supported Visualisations.....	42

System Architecture

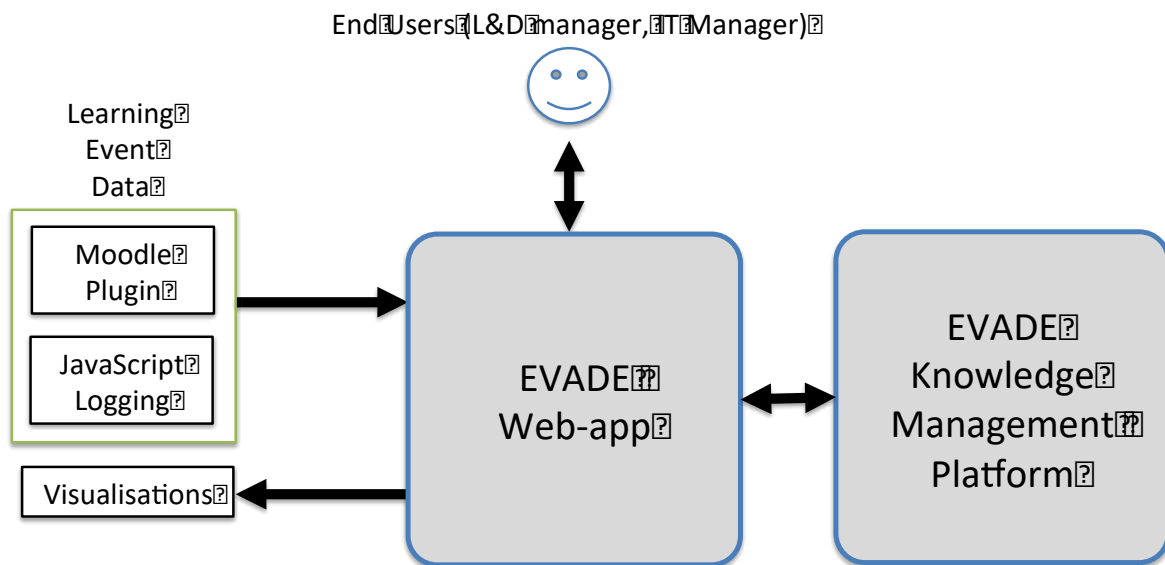


Figure 1. EVADE Overall Architecture

The overall architecture is broken into two core components. Firstly the EVADE web-app as covered in this document forms the user interface for the end users and the also manages the collection of learning events and the embedding of visualisations. The Knowledge Management Platform (KMP) address the storage of the gathered data as well as flexible querying across this data and clustering algorithms. The communication between the web-app and the KMP is via JSON messages sent over HTTP. The exact message format is defined in the API documentation.

Data Design

The data handled by EVADE is in JSON format. A complete specification of this format can be found at <http://www.json.org/>. The benefits of using JSON are:

- Platform agnostic
- Libraries available in virtually all programming languages (see <http://www.json.org/>)
- Native support in web browsers (see Browser JSON Parsing Support)
- Increasing popularity of JSON APIs (<http://java.dzone.com/articles/streaming-apis-json-vs-xml-and>).
- The use of JSON in the Tin Can/Experience API (http://www.adlnet.gov/wp-content/uploads/2013/05/20130521_xAPI_v1.0.0-FINAL-correx.pdf)

As a format JSON has five primitive types and a **null** value:

- String
- Number: Double precision floating-point number.
- Boolean: true or false.

- **Array:** A non-type specific list of elements.
- **Object:** A map consisting of String keys and values of any JSON type.

An example of a Tin Can/Experience API JSON message is given below:

```
{
  "id": "12345678-1234-5678-1234-567812345678",
  "actor": {
    "mbox": "mailto:xapi@adlnet.gov"
  },
  "verb": {
    "id": "http:\\\\adlnet.gov\\expapi\\verbs\\created",
    "display": {
      "en-US": "created"
    }
  },
  "object": {
    "id": "http:\\\\example.adlnet.gov\\xapi\\example\\activity"
  }
}
```

Components Overview

Core Web Application

Platform

Linux (Ubuntu 10.04 64-bit)

Technologies Used

Java 1.6

Tomcat 7.0.40

MySQL 5.1

MariaDB Java Client Library 1.1.5 (used instead of the GPL MySQL connector)

MongoDB 2.4

Groovy 2.0

Grails 2.2.1

Grails Plugins:

hibernate, jquery, resources, geb, rest, cache, cache-ehcache, cors, spring-security-core

Component Dependencies

- EVADE KMP (if used)

Development Environment

The web-app was developed in Eclipse Indigo Service Release 2 (3.7.2) running on OS X 10.8.4 with the following plugins:

- Grails IDE 3.2.0 (aka Groovy/Grails Tool Suite (GGTS) installed from Eclipse Marketplace)
 - Groovy Eclipse Plugin 2.8.0
 - Groovy Compiler 2.0 Feature

Description

The EVADE Web-app is written primarily in Groovy using the Grails web-application framework. It is a dynamic web-application that uses a MySQL database to store the state of the web-app (excluding captured event data). The purpose of this component is to:

- Allow end users to configure APIs to capture data, apply filters to this data, apply queries to the filtered data, configure locations where visualisations are can be embedded, and allow configuration of the data and queries that back the embedded visualisations.
- Provide HTTP APIs that allow the capture of JSON with diverse data formats.
- Store and filter captured data to a separate data storage component.
- Perform queries on the data storage component to generate visualisations.
- Generate visualisations that can be embedded in webpages.

Web-app Architecture

The web-app is designed along best practices for Grails app development. This breaks the app into four types of components in a Model – View – Controller – Service format. This application structure provides clean separation between domain models, presentation, request handling, and application logic respectively. These component types are described below:

- **Models:** The data schema for the web-app, i.e. entities that are persisted to the database and the relationships between them, e.g. User, Visualisations, Embeds, etc.
- **Views:** The front-end presentation of Models using HTML and templates.
- **Controllers:** Handle requests to the web-app and prepare responses based on Models and Services that are rendered using Views.
- **Services:** Core application logic that is used by Controllers and other Services.

The high-level interaction between these components is shown in Figure 2 below.

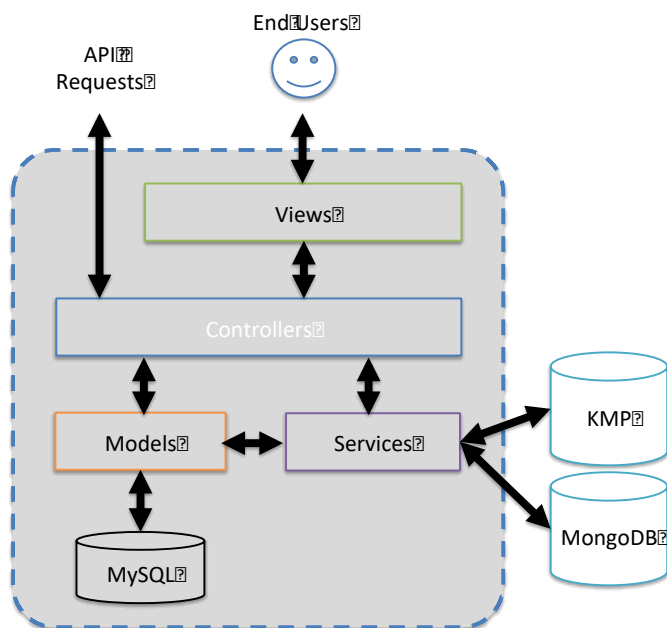


Figure 2 Grails web-app high-level architecture

The main interactions between the specific Controllers and Services in the web-app are shown below in Figure 3. For clarity the Views and domain Models have been omitted from the diagram.

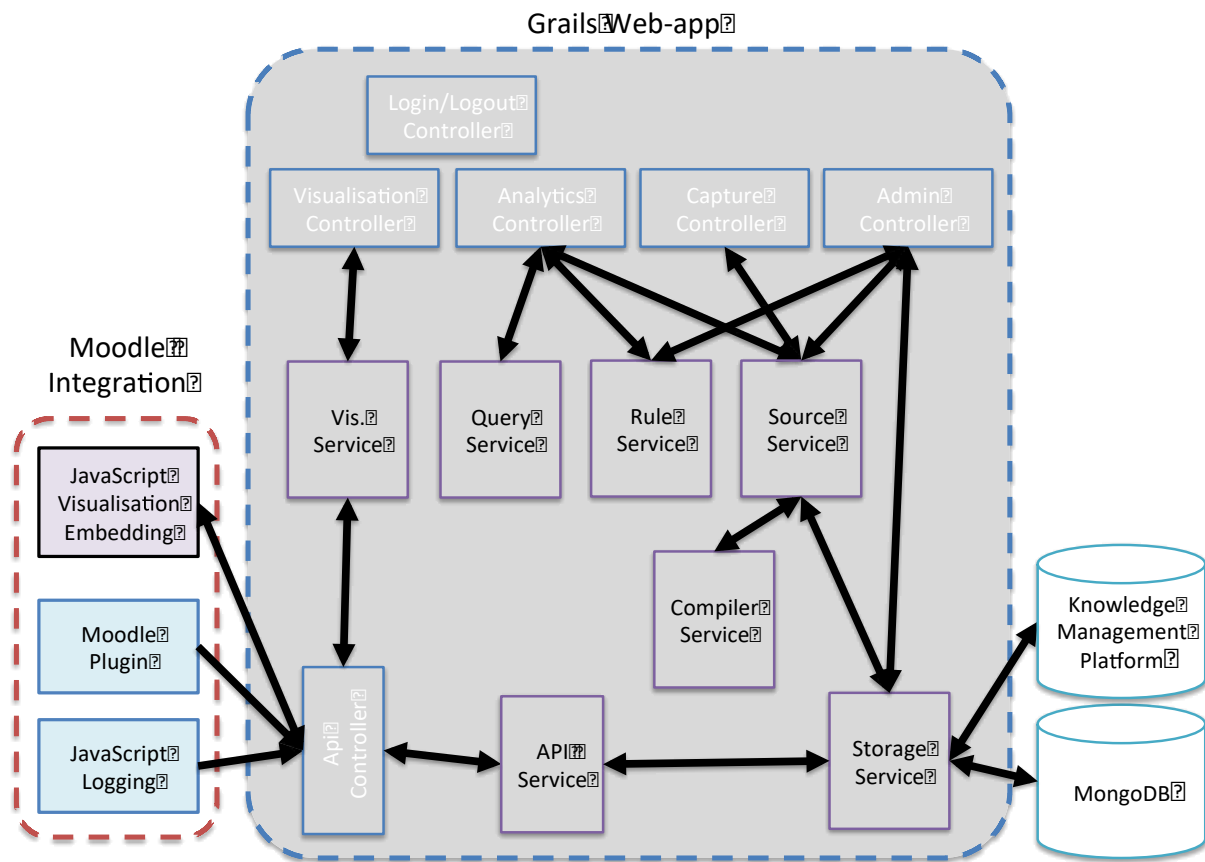


Figure 3. Controller and Service Interaction in the web-app

Security and Authentication

The Spring Security Plugin (spring-security-core) (<http://grails.org/plugin/spring-security-core>) is used to provide authentication and ensure secure access to the web-app and the APIs.

Security

The web-app uses a role-based security model with access to controllers and particular actions being restricted by role. By default access is forbidden to all controllers and must be explicitly granted on a per controller or action basis. The default roles are created in BootStrap.groovy and are:

- **ROLE_USER**: A standard user that can configure visualisations.
- **ROLE_SUPER_USER**: **ROLE_USER** + can create capture APIs, filters, embeds, and queries.
- **ROLE_ADMIN**: **ROLE_SUPER_USER** + can manage users, roles, and user groups.

The role hierarchy is **ROLE_ADMIN** > **ROLE_SUPER_USER** > **ROLE_USER** and is configured in Config.groovy

Controllers/actions are secured with annotations. E.g.

```
@Secured(['ROLE_SUPER_USER'])
class AnalyticsController {...}
```

Session timeout can be configured like a standard web-app by editing 'src/templates/war/web.xml'

Authentication

The web-app uses three types of authentication.

- Session based authentication using a username and password for normal web-app usage.
- HTTP Basic Authentication for embedding visualisations.
- Custom HTTP Authentication for APIs that uses a different api key for each API or a username and api key parameters for JSONP requests (see ApiController).

For normal web-app access, controllers and actions can require specific authentication requirements such as:

- IS_AUTHENTICATED_ANONYMOUSLY: The user has not logged in, i.e. any user.
- IS_AUTHENTICATED_REMEMBERED: The user previously logged in but the session has expired and a 'remember me' cookie is present.
- IS_AUTHENTICATED_FULLY: The user has fully logged in.

Example authentication requirement:

```
@Secured(['IS_AUTHENTICATED_FULLY', 'ROLE_SUPER_USER'])
class AnalyticsController {...}
```

All passwords are stored in the database as hashed values using the SHA-256 algorithm.

Note: At present security hasn't been implemented for the data retrieved for visualisations through requests to 'visDataBatch' in the ApiController.

File Structure

The following is the file structure of the project, explanations are shown in bold.

```
.
├── grails-app
│   ├── conf      (Configuration)
│   ├── controllers
│   ├── domain    (Models)
│   ├── i18n      (Internationalization of messages)
│   ├── migrations (Not currently used)
│   ├── services
│   ├── taglib     (Custom tag libraries for use in GSP views)
│   ├── utils      (Not currently used)
│   └── views
├── lib            (Not used, dependency management used instead)
├── scripts        (Not used)
├── src            (Non grails classes)
└── groovy
```

```

├── java
│   └── templates
├── target      (Build generated content)
├── target-eclipse (Build generated content)
├── test        (Testing classes)
│   ├── functional
│   ├── integration
│   └── unit
├── web-app     (Skelton of the web-app that is built)
│   ├── META-INF
│   ├── WEB-INF
│   ├── css
│   ├── images
│   │   └── skin
│   └── js

```

Configuration

The web-app is configured like a standard Grails application. The key config files are:

```

└─── grails-app
    ├─── conf
    │   ├─── ApplicationResources.groovy (Grails resource bundles)
    │   ├─── BootStrap.groovy (Init code, role creation)
    │   ├─── BuildConfig.groovy (Build and plugin dependency mangagement)
    │   ├─── Config.groovy (Main config options)
    │   ├─── DataSource.groovy (MySQL Database config)
    │   ├─── UrlMappings.groovy
    │   ├─── WebXmlConfig.groovy
    │   ├─── ehcache.xml (Cache config used by hibernate)
    │   ├─── hibernate
    │   └─── spring
    │       └─── resources.groovy

```

Java Web-app config can also be found in:

```
├── src
│   ├── templates
│   │   ├── war
│   │   │   └── web.xml
```

Dependency Management

All dependent libraries and plugins are resolved using Maven and are declared in `BuildConfig.groovy`. For this reason the libraries aren't bundled with the code but instead are downloaded the first time the project is built. For further details see (<http://grails.org/doc/2.2.1/guide/conf.html#ivy>). The one exception to this is the MariaDB java client library that is in the `lib` directory as there was no Maven source for it.

Testing

Testing where implemented makes use of functional test using:

- Geb 0.9.0
- Selenium 2.27.0
- HtmlUnit

```

See // vis = [{url:'http://example.com'},...];
var embeds = [];
$("#demo-table td").each(function(i,element) {
    vis.forEach(function(v,j) {
        if(!embeds[j]){
            embeds[j] = [];
        }
        embeds[j].push({match:$(element).text(), target:element});
    });
});

vis.forEach(function(v,j) {
    EVADE.embedVisBatch(embeds[j], v.url, {append:true, width:60,
height:60});
});
  
```

Known Issues for a note on testing.

Installation and Deployment

- Java 1.6 (untested on 1.7)
- Apache Tomcat 7.0.40 (untested on higher versions but it should work)
 - Forwarding through Apache Http Server is supported and tested for access through port 80. Tested with mod_jk.
 - In catalina.sh the following settings work but may need to be increased for larger deployments
 - export JAVA_OPTS="-Xms128m -Xmx768m -XX:PermSize=128m -XX:MaxPermSize=256m"
- MySQL 5.1 or above
 - A database and user should be created that match the settings in DataSource.groovy
 - The user should have permissions to create and update tables.
 - All tables for the domain models will be automatically created on the first run of the app.
 - The database should be InnoDB and have a UTF-8 character set.
- MongoDB 2.4
 - Requires a database to be created matching the name of 'datastore.mongo.database' in Config.groovy.
 - The server running Mongo should be defined in 'datastore.mongo.server' in Config.groovy

Following the above setup the EVADE war file should be deployed in Tomcat. Check the Tomcat logs for any errors. There is by default one admin user created with the username 'admin' the default password can be configured in BootStrap.groovy.

Open Source Licenses

The following open source licenses are used within the web-app:

	Apache License v2	LGPL	Eclipse Public License	BSD License
Groovy	X			
Grails	X			
CORS plugin	X			
Cache Plugin	X			
jQuery Plugin	X			
Resources Plugin	X			
Spring Security Plugin	X			
Resources plugin	X			
Mongo Java Driver	X			
Drools	X		X	X
MariaDB Client Java Library*		X		
HttpClient	X			
Javassist	X	X		
Tomcat	X			
Guava libraries	X			
Resources plugin	X			
Selenium	X			
Apache Commons Pool	X			
Hibernate		X		
Log4J	X			

Figure 4. Table of Open Source licenses used

* This JDBC connector is used instead of the MySQL one as it is LGPL as opposed to the MySQL connector, which is GPL.

Online Documentation

- Groovy 2.0
 - User Guide: <http://groovy.codehaus.org/User+Guide>
- Grails 2.2.1
 - Documentation: <http://grails.org/doc/2.2.1/>
- Spring Security Plugin

- <http://grails.org/plugin/spring-security-core>
- JBoss Drools
 - User Guide: <http://docs.jboss.org/drools/release/5.5.0.Final/drools-expert-docs/html/index.html>
 - API docs: <http://docs.jboss.org/drools/release/5.5.0.Final/knowledge-api-javadoc/index.html>
- JavaAssist
 - <http://www.csg.ci.i.u-tokyo.ac.jp/~chiba/javassist/>
- MongoDB 2.4
 - <http://docs.mongodb.org/v2.4/>

Grails Models

The Models in the web-app represent the classes that are used in the web-app that are persisted to the MySQL database.

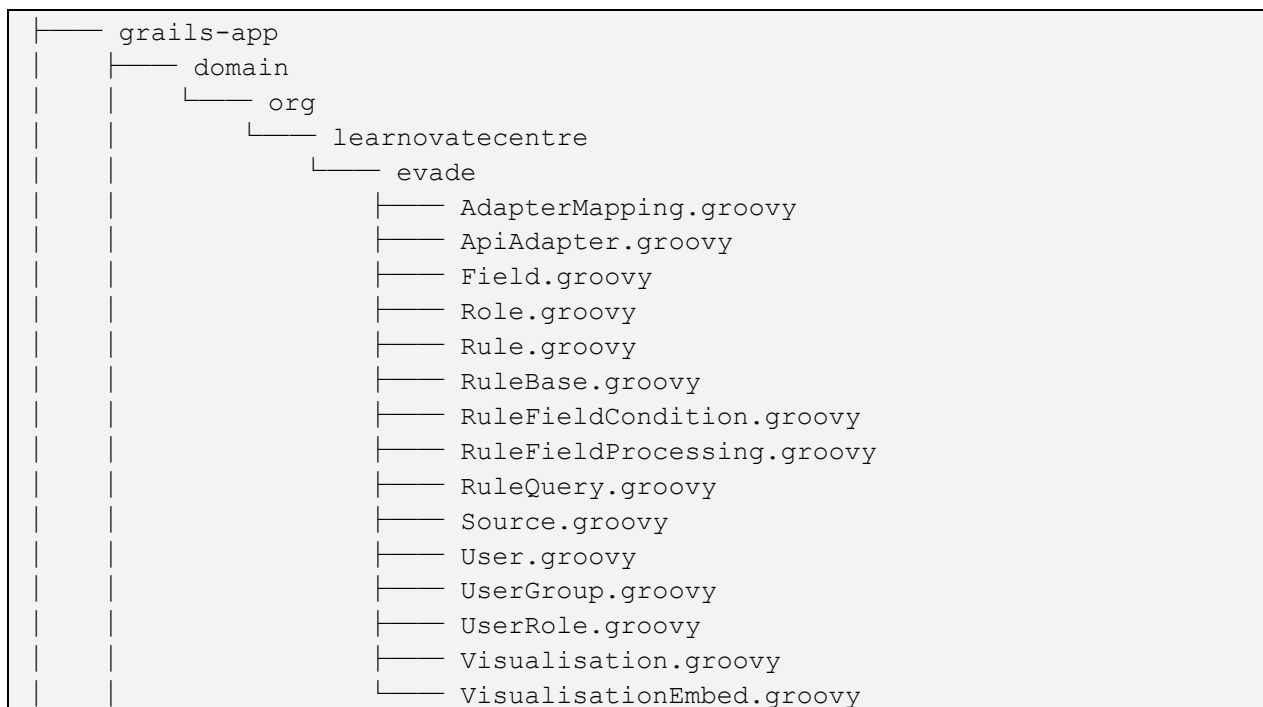
Technologies Used

- Groovy 2.0
- Grails 2.2.1

Description

The following are the domain models used in web-app. The class diagram showing the relationships between the models is shown in Figure 5. The class diagram includes some properties that not serialized to the database, the classes can be inspected to see which properties are serialized.

Domain Model Classes



UML Class Diagram

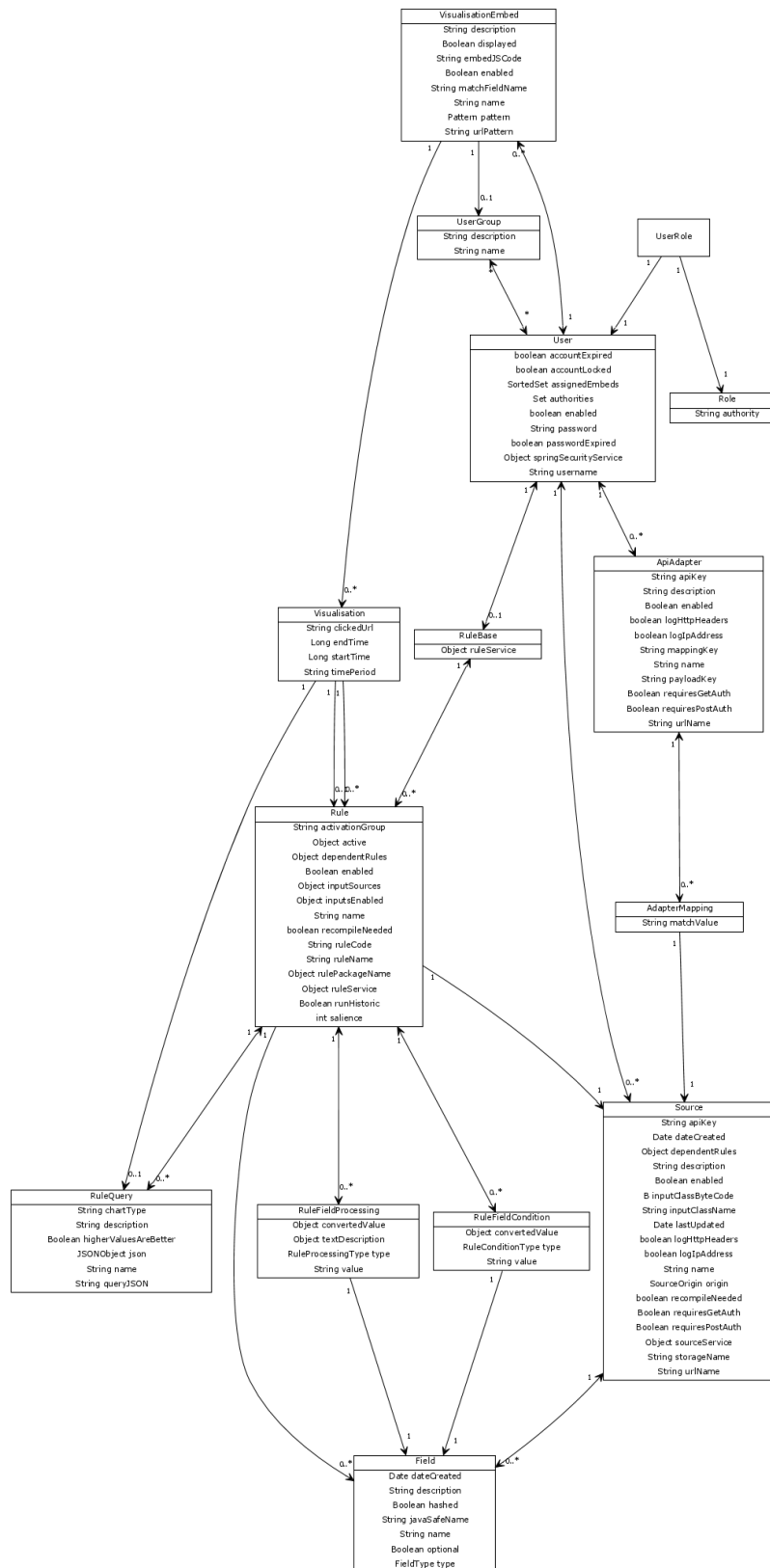


Figure 5. UML Class Diagram of Domain Models

User

This model represents a user of the web-app.

UserGroup

This model represents a group of users. A user can be in any number of user groups. Users in the same group can configure the same Visualisation Embeds.

Role

A role that a user can have. This can be either:

- User (ROLE_USER) : Can configure Visualisation Embeds
- Super user (ROLE_SUPER_USER) : User + can create APIs, filters, queries, and embeds.
- Admin (ROLE_ADMIN) : Super user + can add/edit users and misc admin functions

Roles are created in Bootstrap.groovy and their hierarchy is defined in Config.groovy.

UserRole

A mapping between users and roles.

Source

A source of data either from an API or the output of a Rule. This class extends ApiTarget although this is only relevant if it's origin value is SourceOrigin.API. A Source has a number of Fields.

Field

A data field within a data Source.

ApiAdapter

This model represents a convenience adapter that allows several data capture APIs (Sources) to be routed through one API URL. An adapter has one or more AdapterMappings that map particular input messages to Sources.

AdapterMapping

A mapping between specific parameters sent to an ApiAdapter and the Source to send the data to.

Rule

A processing rule, also referred to as a filter, that outputs optionally processed data to a Source if the configured conditions all hold true. A Rule can include/exclude certain fields from an input Source as well as applying RuleFieldConditions and RuleFieldProcessings.

RuleFieldCondition

A condition that is applied to a Field by a Rule. Valid types of conditions for each FieldType are available in RuleConditionType.

RuleFieldProcessing

Some processing that is applied to a Field by a Rule. Valid types of processing are defined in RuleProcessingType.

RuleQuery

A query that can be performed on the data stored in the output Source of a Rule. A RuleQuery also defines what sort of visualisation can be created from the resulting data.

RuleBase

A collection of Rules that is owned by a User.

VisualisationEmbed

A location within a web-page where a group of Visualisations are to be embedded.

Visualisation

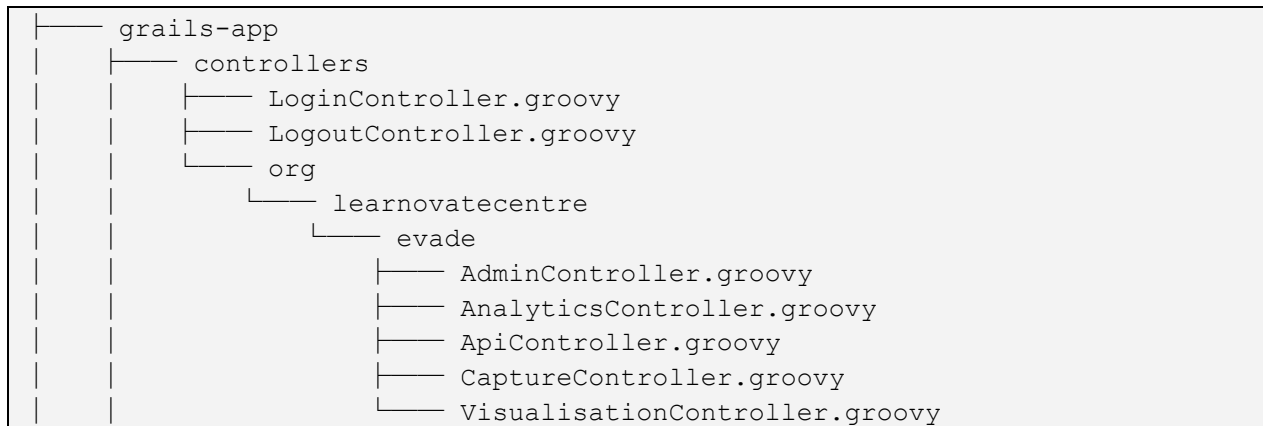
A visualisation that appears in a VisualisationEmbed that is generated from the data resulting from a RuleQuery on a Rule's output Source.

Grails Controllers

The controllers within a Grails app manage requests to the web-app and determine what response to serve including the graceful handling of errors.

In general controller URLs are mapped as `/$controller/$action?/$id?` (see `UrlMappings.groovy`)

Controller File List



Technologies Used

- Groovy 2.0
- Grails 2.2.1

LoginController

This class is generated by and used by the Spring Security Plugin to manage user logins.

Path: `/login/`

LogoutController

This class is generated by and used by the Spring Security Plugin to manage user logouts.

Path: `/logout/`

AdminController

This controller allows admin users to manage the creation of users and the assignment of roles. An interface is also provided to configure logging at runtime.

Requires role: admin

Path: `/admin/`

AnalyticsController

This controller allows super users to add filters (rules) to Sources captured through APIs.

Requires role: super user

Path: /analytics/

ApiController

This controller does not generate HTML but instead acts as a HTTP JSON API to allow configured Sources and ApiAdapters to capture data, and to allow data for Visualisations to be retrieved. Authentication is configured on a per Source basis.

Note: Hibernate sessions are disabled for /api/post for performance reasons. See FilteredOpenSessionInViewInterceptor and *osiv.excludes* in Config.groovy. As a result grails lazy loading and query caching won't work in this method.

Path: /api/

CaptureController

This controller allows super users to configure Sources to capture Api data through the ApiController.

Requires role: super user

Path: /capture/

VisualisationController

This controller allows super user to create/edit Visualisation Embeds and also allows users to configure these embeds.

Requires role: user/super user

Path: /visualisation/

Miscellaneous Controller Notes

The convenience mixin ControllerUtils.groovy is used in some of the controllers to send JSON responses.

To allow UI feedback during long running tasks the AjaxStream class can be used to stream responses back to a web browser. This class must be used in conjunction with the modified jQuery Stream plugin (jquery.stream-1.2-mod.js). This functionality is used in the AnalyticsController to stream updated to when filtering historic data that may take a long time to complete.

Grails Views

The views in the web-app represent the HTML presentation of the domain models and the interface for the user to interact.

Description

As in a standard Grails app the view directories correspond to their respective controller names, as well as directories for shared views and a directory for SiteMesh layouts. The **rule** directory is the exception to this as it does not correspond to a controller but instead contains GSPs used to generate Drools Rules in the AnalyticsService.

As is standard in Grails all views are named according to their corresponding controller action name, e.g. the URL “/capture/show/1” uses the view “/views/capture/show.gsp”. Each view generates a whole HTML page except for templates that generate HTML fragments. All templates start with an underscore (“_”) in their file name.

A custom Grails TagLib is used in several views. The code for the custom tags can be found in:

```

|— grails-app
|   |— taglib
|       |— evade
|       |— UtilsTagLib.groovy
  
```

Views File List

```

|— grails-app
|   |— views
|       |— admin
|       |— analytics
|       |— api
|       |— capture
|       |— layouts (SiteMesh layouts)
|       |— login
|       |— rule (GSPs to generate Drools Rules)
|       |— shared (views shared across controllers)
|       |— visualisation
  
```

Technologies Used

- Groovy 2.0
- Grails 2.2.1
- GSP (Groovy Server Pages)
- SiteMesh (as used by Grails)
- HTML
- JavaScript
- jQuery 1.8.3

- D3.js v3
- CSS
- Twitter Bootstrap 2.3.2

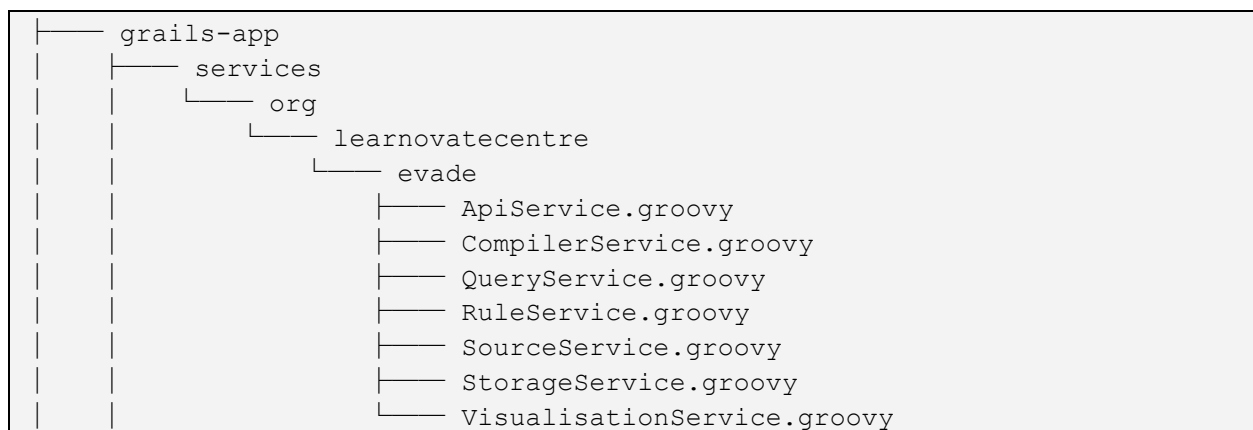
Tag libraries

Some utility GSP tags are defined in:

```
|— grails-app
|   |— taglib
|       |— evade
|           |— UtilsTagLib.groovy
```

Grails Services

In a Grails app the bulk of the application logic is found in the services. By default all services are transactional and are rolled back if the service call throws an exception. The ApiService and StorageService are not transactional as they don't create or update any domain models and there is additional overhead associated with creating a transaction for each request.



Technologies Used

- Groovy 2.0
- Grails 2.2.1
- JBoss Drools 5.5.0
- JavaAssist (part of Grails)

Component Dependencies

- StorageService
 - MongoDB
 - KMP (if used)

ApiService

This service is primarily used by the ApiController to check data submitted to APIs and also to execute rules on the new data.

CompilerService

This service is used to compile a Java class that represents a Source and can be used by JBoss Drools. When data is received for an API Source an instance of the generated class is created and inserted into the working memory where it will trigger any filtering or processing rules (see RuleService). See also SourceClassCompiler for actual compilation that is performed using JavaAssist.

QueryService

This service creates and updates RuleQueries

RuleService

This service manages the creation and deletion of Rules as well as the generation of Drools rule code for each Rule, and the management of working memories that execute the rules. The filtering and

processing of API data from Sources is implemented using a Drools rulebase per user. As users add and remove filters, rules are generated (using the rule views), compiled, and added to the user's Drools working memory. When new data is received by an API it is inserted into the working memory and any matching rules will execute. Each rulebase has a default that causes any inserted fact to be stored by the StorageService.

SourceService

This service manages the creation, updating and deletion of Sources.

StorageService

This service manages the storage of data received through Sources either from APIs or from Rules. The Data is stored to either the MongoDB data storage or to the KMP component.

VisualisationService

This service manages the creation, updating and deletion of VisualisationEmbeds and Visualisations.

CacheService

This service enables the caching of values primarily used during API calls to prevent MySQL database hits. Caches are configured in Config.groovy.

Non-grails Classes

The following are classes used by the web-app that do not fit into the normal MVCS structure of a Grails app. These typically are utility classes and enums.

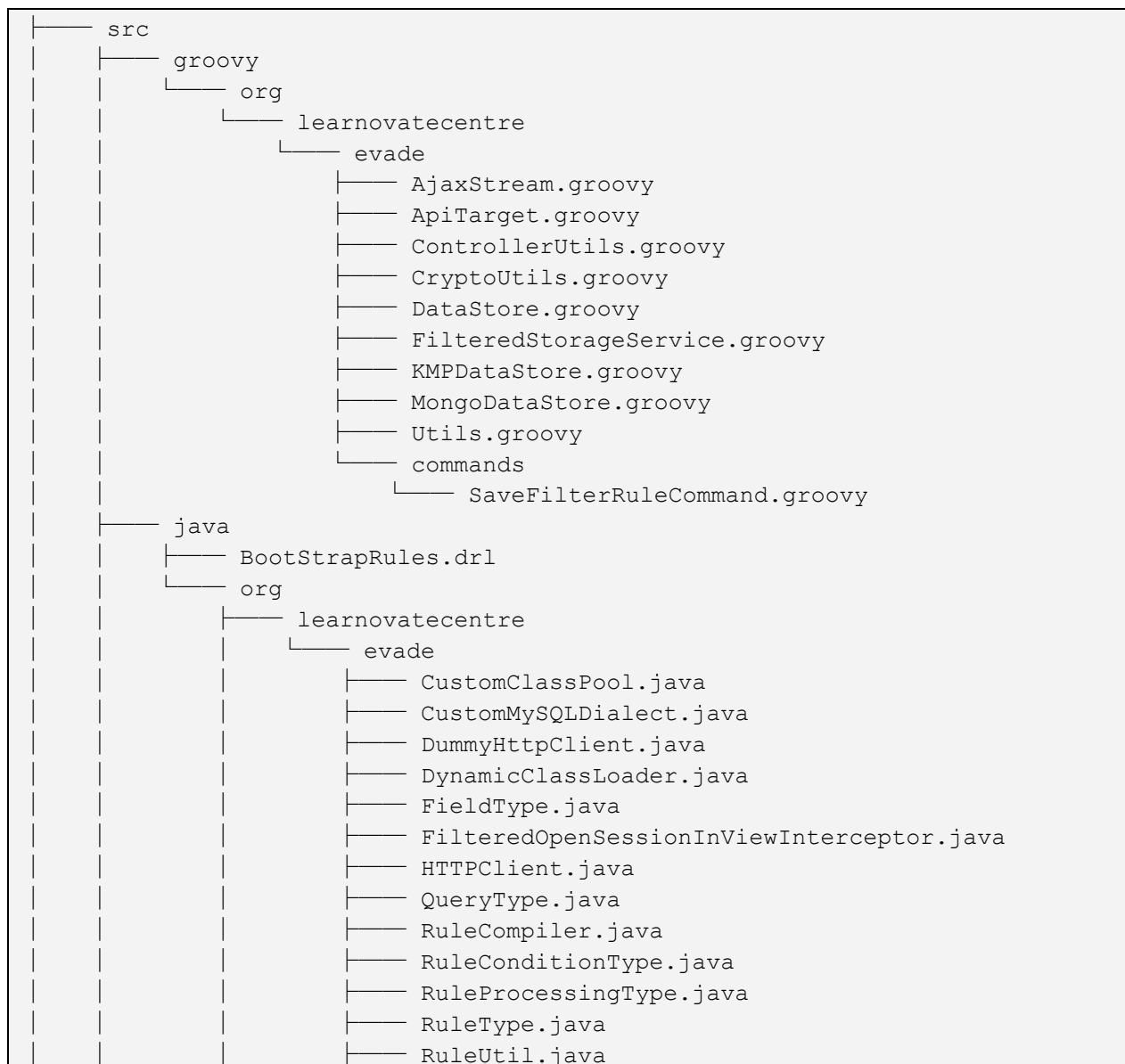
Technologies Used

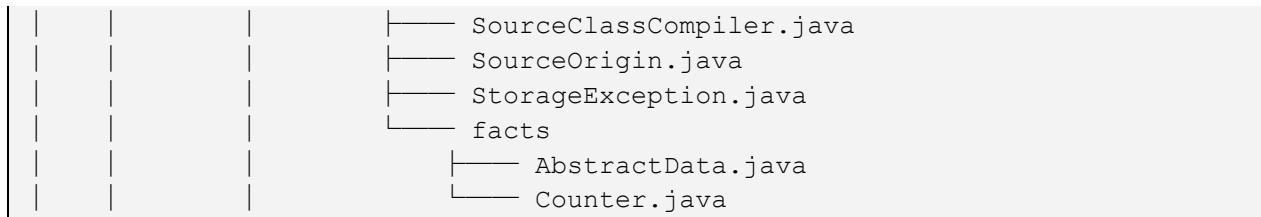
- JavaAssist (as part of Grails)
- Apache HttpClient 4.2.5 (Used by HTTPClient)
- Google Guava 13.0.1 (Used by the RuleCompiler and DynamicClassLoader)

Component Dependencies

- KMP (if used)

Non-Grails Classes File List





AjaxStream

This class represents a simple streaming response sent to a HTTP client. It can be used to send intermittent updates to a client for long running processes. It requires client support and uses the modified jQuery Stream plugin in the browser. It is used when new rules are processing historic data as this can take a long time (see Analytics Controller).

ApiTarget

This is an abstract class representing classes that can be the target of an API. Source and ApiAdapter both subclass this class.

ControllerUtils

This class is intended to be used as a mixin for Controllers. It provides convenience methods to send JSON responses.

CryptoUtils

This class provides methods to generate SHA-256 hashes. It is used to generate API keys.

DataStore

This is an interface that describes data stores. It is implemented by `KMPDataStore` and `MongoDataStore`. Each implementing class should support queries made in the Internal Query Format as detailed in the References section.

FilteredStorageService

This class extends the `StorageService` and only allows classes to be stored that match the specified collection of class names.

KMPDataStore

This class provides an interface to the Knowledge Management Platform using a JSON messages over HTTP. See the API documentation for exact message formats. The communication is achieved using the HTTPClient class which in turn uses the Apache HttpClient library. This class also implements features that are not natively supported by the KMP such as calculating averages, summation, sorting, and calculating quantiles. Configuration options for this class including endpoint URLs are read from `Config.groovy` in the 'datastore' section.

MongoDataStore

This class provides an interface to a MongoDB database that can act as a stand in replacement for the KMP. This class communicates directly with a MongoDB instance as opposed to using JSON messages over HTTP. The server hosting the MongoDB instance is configurable in `Config.groovy` under

'datastore.mongo.server', as is the Mongo database name in 'datastore.mongo.database'. This class also implements features that are not natively supported by MongoDB such as calculating quantiles.

Utils

A class for utilities. Notably 'dumpStack(...)' that returns a String containing the stack trace of a Throwable that only includes lines matching a specified package name.

SaveFilterRuleCommand

A Grails Command object that is used when saving a Rule.

CustomClassPool.java

A JavaAssist ClassPool that uses DynamicClassLoader.

CustomMySQLDialect.java

Used in DataSource.groovy to define MySQL tables as being InnoDB utf-8.

DummyHttpClient.java

Used in testing the KMPDataStore instead of actually making connections using the HttpClient.

DynamicClassLoader.java

A class loader that can load java classes representing Sources from the MySQL database.

FieldType.java

An enum of the possible types a Field in a Source can have.

FilteredOpenSessionInViewInterceptor.java

This class extends the GrailsOpenSessionInViewInterceptor and adds the ability to not open a Hibernate session for particular requests. This is used to prevent sessions being created for API calls to reduce the overhead for storing data. This class is configured in Config.groovy with the values of *osiv.excludes*. The class overrides GrailsOpenSessionInViewInterceptor by replacing the bean in conf/spring/resources.groovy.

HttpClient.java

A multi-threaded interface to send HTTP GET and POST requests. This is based on Apache HttpClient and is used to communicate with the KMP.

RuleCompiler.java

A class to compile Rules into a JBoss Drools KnowledgeBase.

RuleConditionType.java

An enum of the condition types that a field condition can have in a Rule.

RuleProcessingType.java

An enum of the processing types that a field processing can have in a Rule.

RuleUtil.java

Utility class used by compiled Drools rules to convert between types safely.

SourceClassCompiler.java

A class to compile classes to represent the data of Source at runtime. Uses JavaAssist.

SourceOrigin.java

An enum of the possible origins or a Source, API or RULE.

StorageException.java

An Exception that can be thrown by the StorageService and Data Stores.

AbstractData.java

An abstract class that is extended by all Source classes that are compiled and represent data that is received by APIs or generated by filter rules.

Counter.java

A counter class that can be used in Drools rules, it is not currently used.

User Interface Design

User Interface Overview

The user interface is designed around catering for two main user types, namely a standard user (e.g. a L&D manager) and a super user (e.g. an IT manager). There is also an administrator user that is equivalent to a super user but with access to user management.

The UI is entirely a web-based interface written in HTML, CSS, and JavaScript. Although the interface is predominantly designed for desktop users it does feature responsive components due to use of the Twitter Bootstrap library. This should make the UI accessible on space constrained and touch devices.

Workflow Examples

The following are workflows for the normal and super users. The workflow for the admin user only involves adding and amending user accounts so it has been omitted.

Super User

1. The user logs into the EVADE web-app

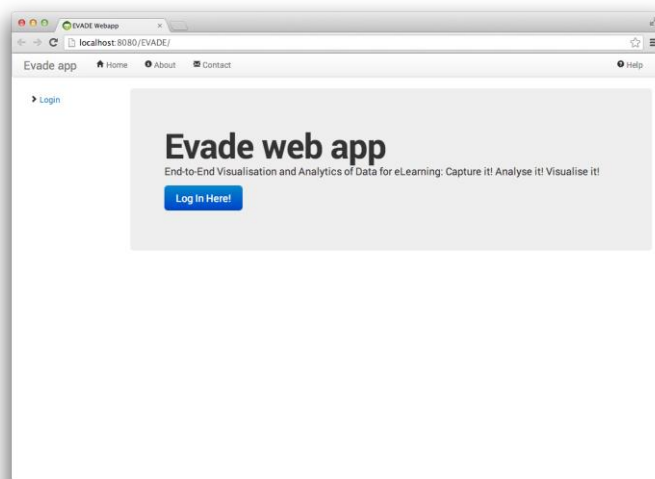


Figure 6. Web-app home page

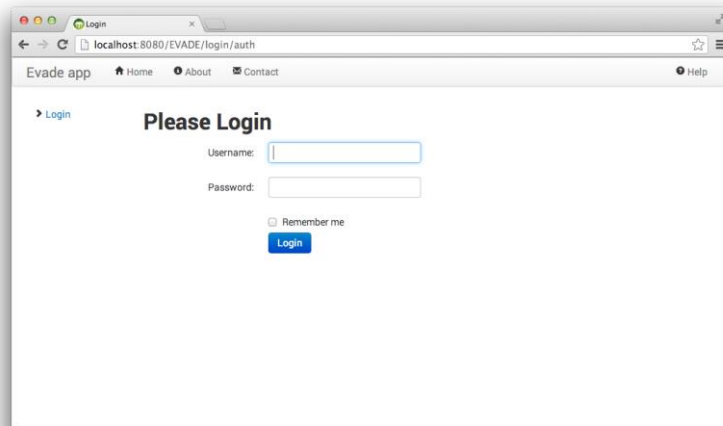


Figure 7. Web-app login page

2. The user views the configured Data Capture APIs and creates a new one.

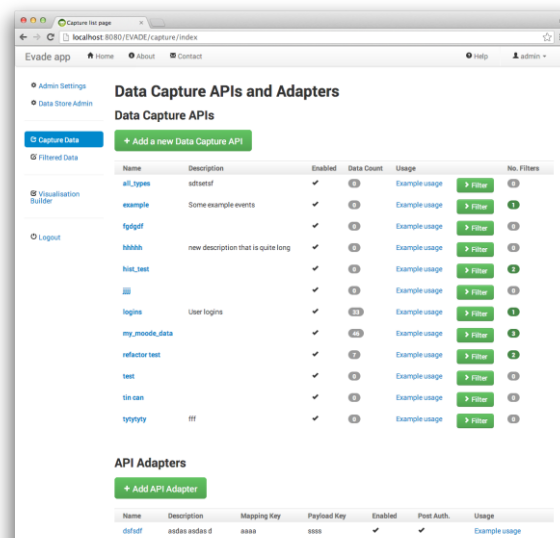


Figure 8. Data Capture API List page

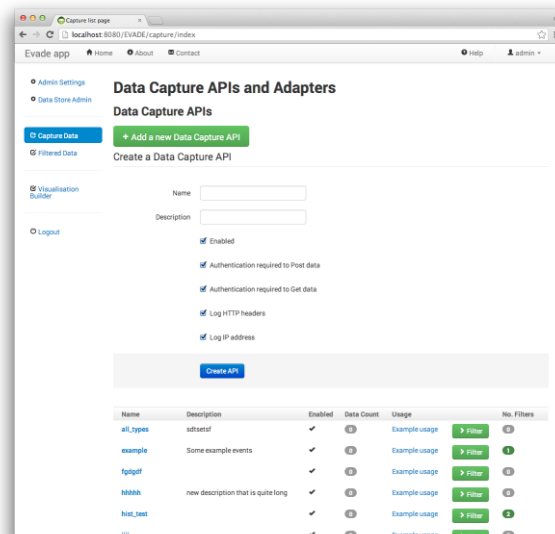


Figure 9. Creating a new Data Capture API

3. The user edits the fields in the data to be captured. Field types correspond to JSON format types, namely a String, a Boolean, a Number, an Array, and a nested JSON Object. Fields must have a name and can optionally have a description. Fields can be made optional if desired. String fields also have the option of being one-way hashed with a salt (using SHA-256) to anonymise data.

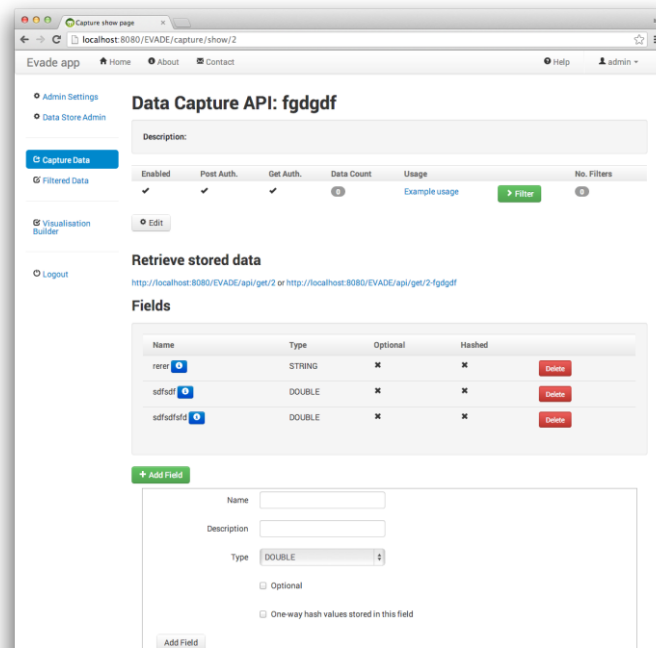


Figure 10. Adding fields to a Data Capture API

- Once all the fields have been added the user can click on 'Example Usage' to get details of how to send data to this newly configured data source and optionally send test data to it. N.B. There is currently no way to remove test data once it has been added.

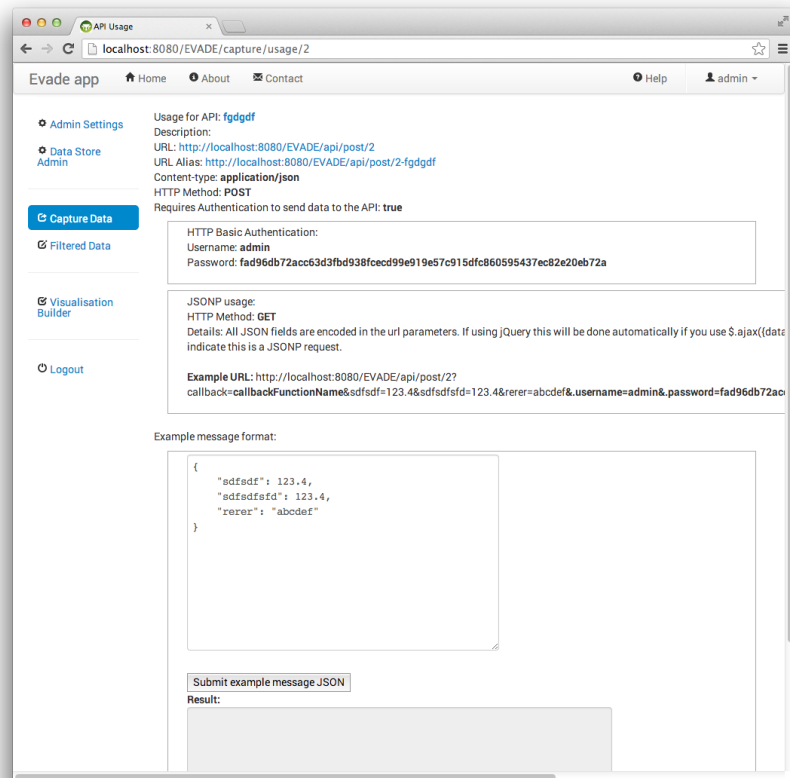


Figure 11. Viewing example usage for a Data Capture API

- Once an API has been created it can be filtered by clicking on 'Filter' and processed into several different filtered data sets. A filter allows certain fields to be included/excluded, conditions to be put on fields, and processing to be performed on data to transform it. In the case where data has already been captured by an API it is possible to run the filter on historic data. If this option is not selected only new data received by the API will have the filter applied.

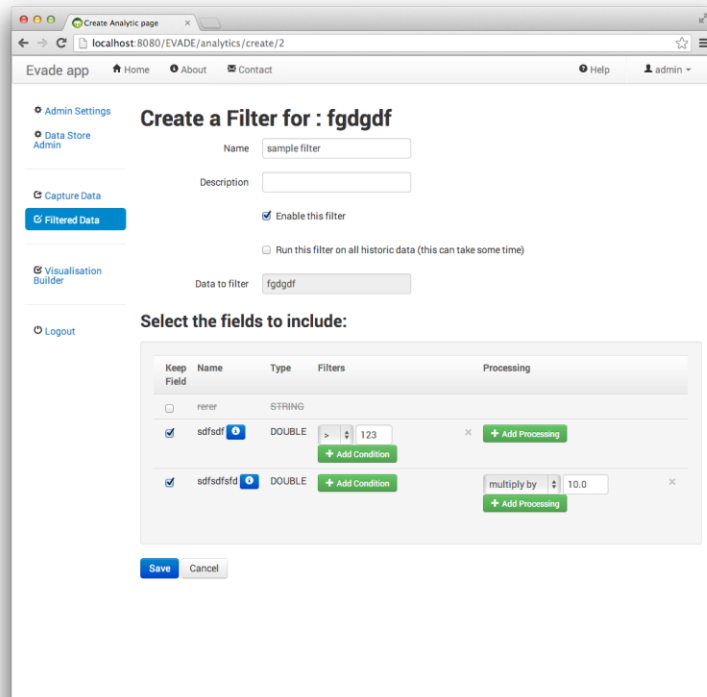


Figure 12. Creating a filter on captured data

- Once a filter has been created the user can create queries on the filtered data. This can be done by clicking 'Add Query' on the filtered data list. When creating a query a **Chart Type** has to be specified that can be either **Text** (plain text of the results), **Bi-level donut** (requires five quartile values and a bullet value), **Two value donut** (requires two numeric values). This is the type of visualisation generated and depends on the query used. Multiple sub-queries can be created in order to generate more complex results.

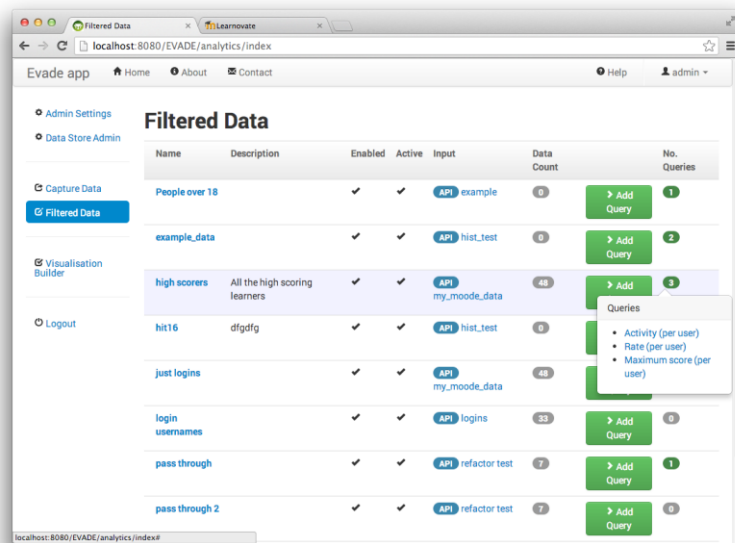


Figure 13. Viewing all filtered data sources

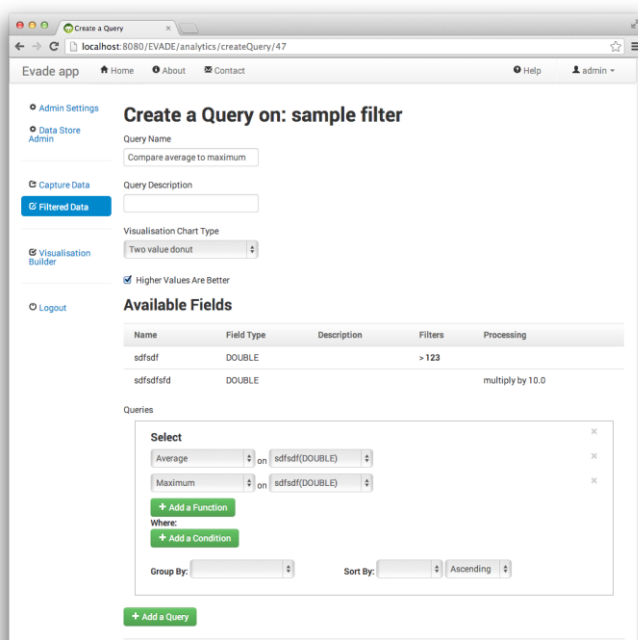


Figure 14. Adding a query to filtered data

- Once at least on query has been defined on some filtered data it is possible to create an embed location to embed the visualisation using the Visualisation Builder. As the embeds will be configured by normal users it is good to give a meaningful name and description to the embed. The **User Group** restricts configuration of the embed to only those users in the chosen user group. User Groups must be configured by an admin user.

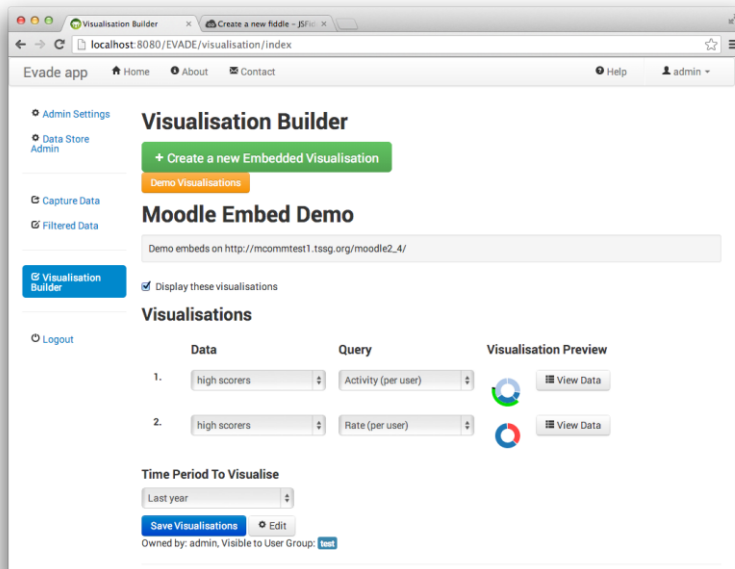


Figure 15. Super User Visualisation Builder view

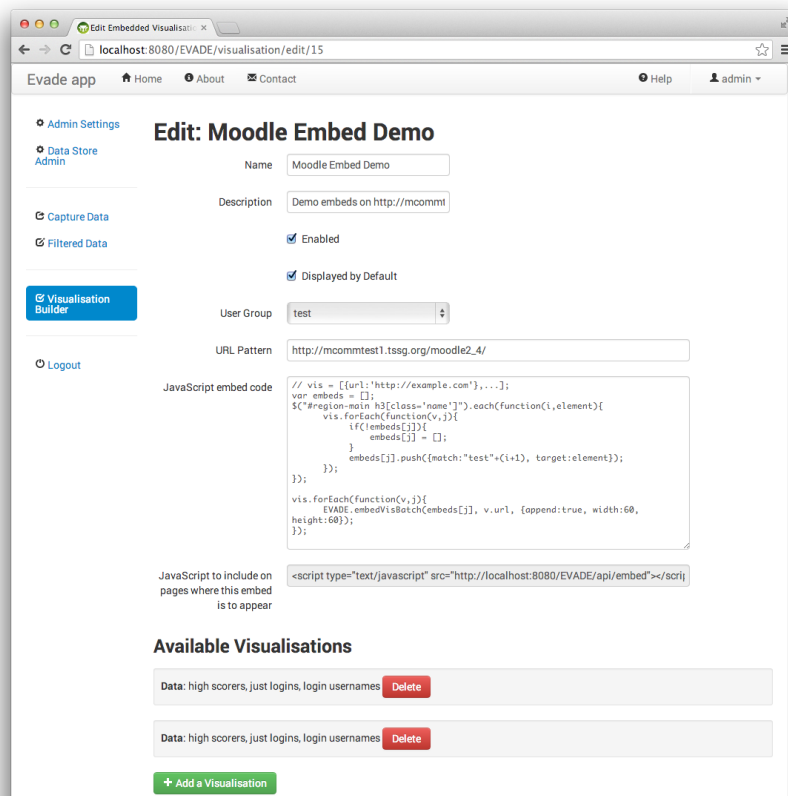


Figure 16. Editing an embed location

- Once an embed has been configured it can be installed into a webpage by including the embed JavaScript file on all pages that will have the visualisations. This script file is always the same URL and can safely be included on all pages, however you will be prompted to authenticate if you are not logged in. An embed must be configured before it will work, this process is shown in the Normal User workflow.

Normal User

- The normal user has a simple workflow where for each visualisation they have to select the data they want to visualise, the query they want to apply, and time period of data to consider. Examples of the visualisation types are shown when selecting different queries. Examples of the visualisation types are shown when selecting different queries.

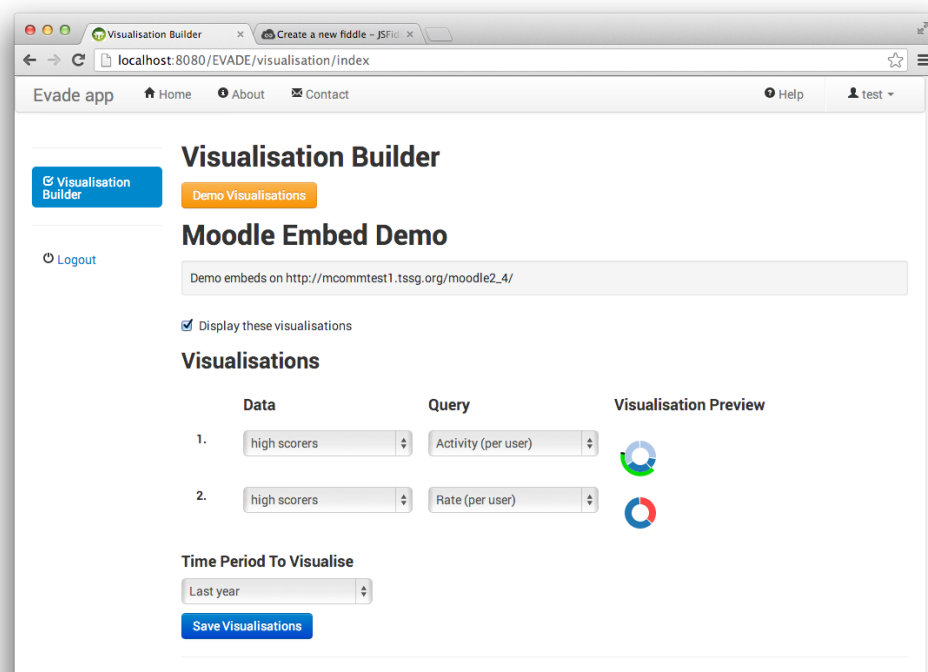


Figure 17. Normal user Visualisation Builder view

- Once embedded visualisations have been saved they can be viewed within the embed location, in this example the visualisations are embedded within a Moodle page.

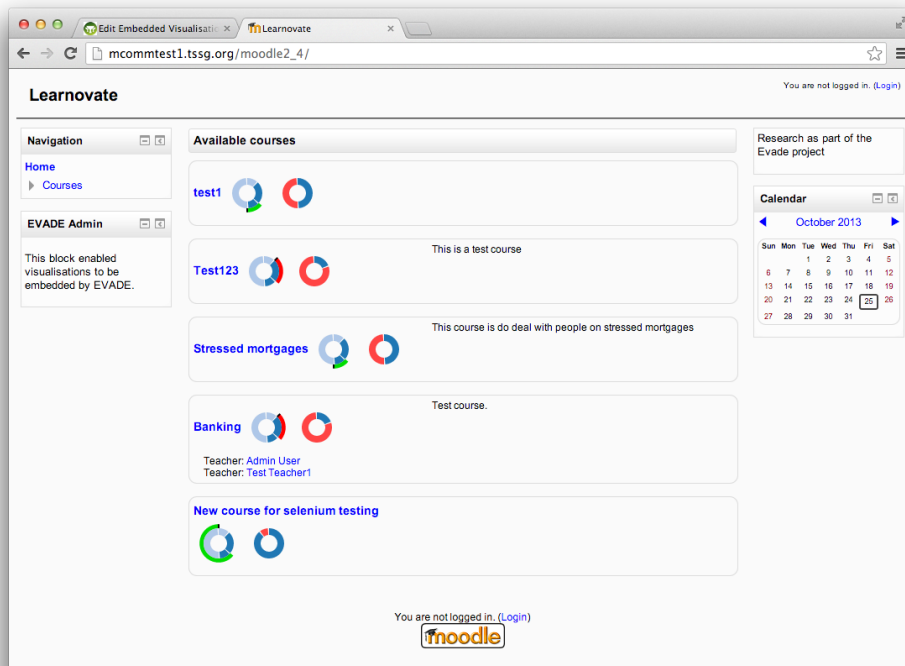


Figure 18. Example visualisation embedded within Moodle

Admin User

The admin user has the same access privileges as a Super User except they can also access the following admin interfaces to manage users and to manage MongoDB data respectively.

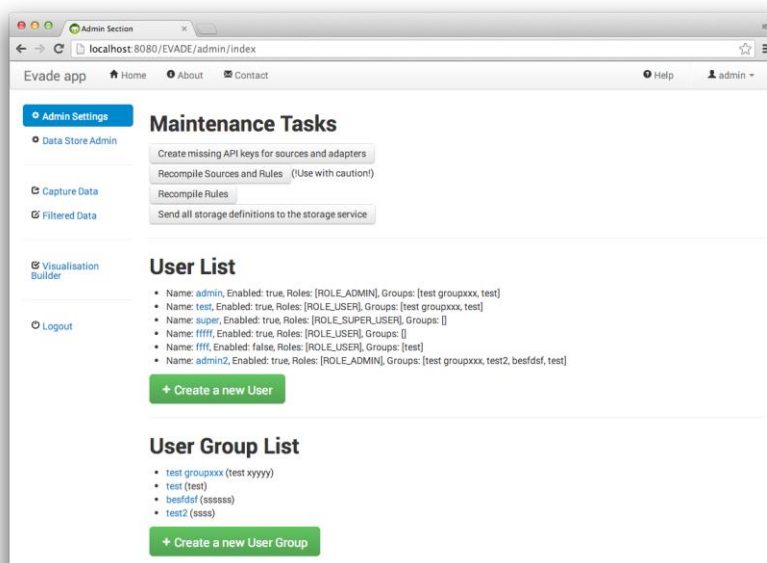


Figure 19. Admin Settings

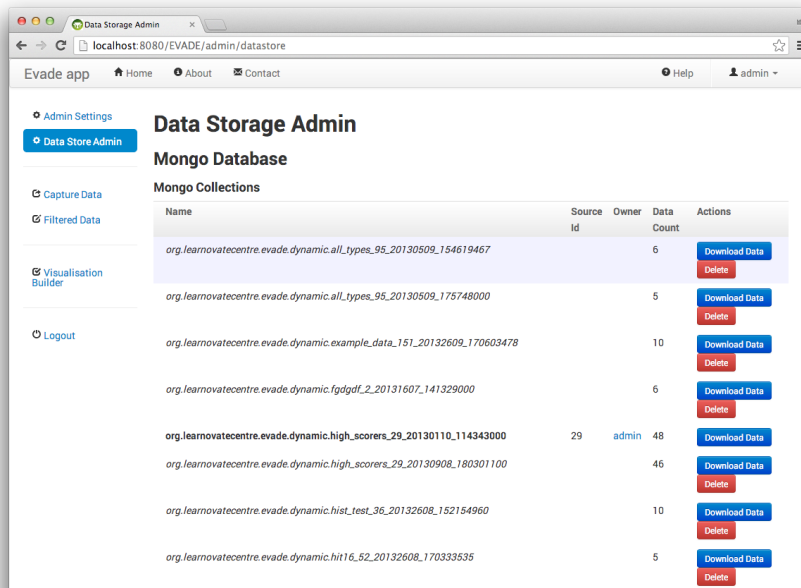


Figure 20. Mongo Data Store Admin

Visualisation Generation

Technologies Used

- HTML
- JavaScript
- JQuery
- D3.js v3 (requires browser SVG support, see Browser SVG Support)

The generation of the visualisations is achieved through the use of D3.js to generate inline SVG visualisations. The advantage of this approach is that the visualisations will display consistently across browser platforms and will scale to arbitrary sizes without pixilation. The D3 code to generate the visualisations is in the file:

```

├── web-app
│   └── js
│       └── visualisations.js
  
```

Whilst the visualisation data is being loaded a clock symbol is shown where the visualisations will appear.

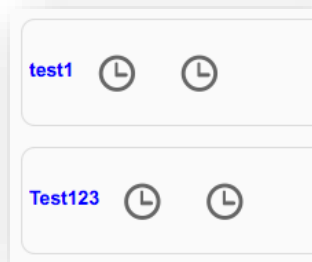


Figure 21. Visualisation loading symbols

If an error occurs due to malformed/incomplete/unavailable data or an unknown visualisation type the following error icon will appear. Check the JavaScript console for further details in this scenario.

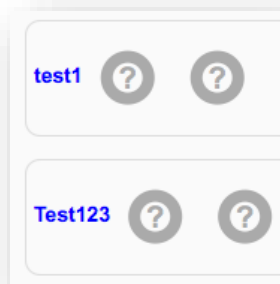


Figure 22. Example visualisation errors

Supported Visualisation

The current supported visualisations are a Bi-level donut chart (used to show activity), a Two-value donut chart (used to show rates), and plain text (used for debugging) as shown below.

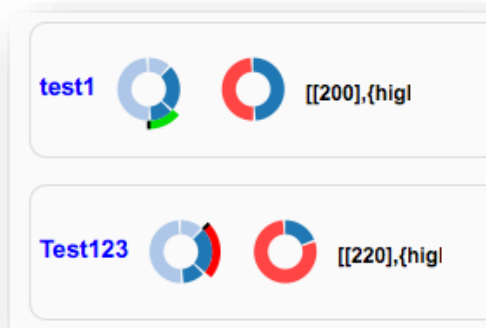


Figure 23. Supported Visualisations

Adding additional visualisations

Additional visualisations can be added by adding D3 code to **visualisations.js** and also by modifying the file below to allow the new visualisation type to be selected.

```

├── grails-app
│   └── views
│       ├── analytics
│       └── _editQuery.gsp
  
```

Visualisation Embedding

The embedding of visualisations has been designed to require a minimal amount of effort for those integrating EVADE into their webpages, such as Moodle.

Embedding Requirements:

- JQuery (1.8.3+ recommended)
- Web browser that supports SVG
- Inclusion of the **/api/embed** JavaScript file.
e.g.

```

<script type="text/javascript" src="http://localhost:8080/EVADE/api/embed"><
/script>
  
```

Embedding Process

1. The **/api/embed** JavaScript is loaded on a webpage.
2. The authentication of the user is checked, if they are not currently logged in to the EVADE web-app they will be prompted for a username and password (HTTP Basic Auth).
3. Any dependencies for the visualisations are loaded e.g. D3.js, visualisations.js.
4. All of the user's configured embeds have their URL patterns compared to the current page (based on the HTTP referer header). Any that match have their embed code returned as part of the JavaScript file and executed.
5. Each embed code should make one or more calls to:
 - a. **EVADE.embedVisBatch(embeds, url, options);**

This JavaScript function is defined in:

```

├── web-app
│   └── js
│       └── visualisations.js
  
```

It takes an array of objects that define the query parameter (if any) and the DOM node where to embed the visualisation, e.g. [{match:"username1", target:node},...]. The **url** parameter is the URL for the visualisation data and points to **/api/visDataBatch/<visualisation-id>** and is retrieved from the **vis** array that is available for each embed code. The **options** parameter is an optional object with the following possible parameters:

- **append:** whether to append the visualisation to the target node or to overwrite its contents (default: true).
 - **width:** the width of the visualisation in pixels (default: 60).
 - **height:** the height of the visualisation in pixels (default: 60).
 - **style:** CSS styles to add to the embedded svg HTML element.
6. Once visualisation data has been retrieved each visualisation is generated using D3 and embedded in the target page's DOM.

Example Visualisation Embed code for an embed

```
// vis = [{url:'http://example.com'},...];
var embeds = [];
$("#demo-table td").each(function(i,element) {
  vis.forEach(function(v,j) {
    if(!embeds[j]){
      embeds[j] = [];
    }
    embeds[j].push({match:$(element).text(), target:element});
  });
});

vis.forEach(function(v,j) {
  EVADE.embedVisBatch(embeds[j], v.url, {append:true, width:60,
height:60});
});
```

Known Issues

- When fields are added or removed from a data capture source it is considered to be a new source of data and all previously stored data is no longer associated with it. This data can still be accessed through the Admin Data Store interface.
- No authentication has been implemented yet for the APIs to retrieve the data for visualisations. The 'visDataBatch' action in ApiController needs to be secured.
- A circular dependency exists somewhere in the app (possibly with the class loaders) that prevents the perm gen from being garbage collected when the app is undeployed from Tomcat. This can lead to the perm gen space becoming exhausted after a few redeployments of the app. Restarting Tomcat resolves this issue.
- The visualisations require browser support for SVG which excludes Internet Explorer 6,7,8 and Android browser < 3.0 . A full compatibility report is given in [Browser SVG Support](#).
- Unit tests and functional tests have not been implemented or are outdated due to the code base changing.
- HTTPS support hasn't been added to the app yet.
- In the UI the links to About, Contact, and Help have not been implemented yet.

References

Internal Query Format

Within the web-app an internal format for data store queries is used that is not specific to either the KMP or the MongoDB data store. This format is a super-set of the features natively supported by either data store. Classes that implement the DataStore interface are expected to handle this format either natively at the database level or in code. The query format is defined as a JSON object as shown below. Each query can contain multiple queries that will be executed sequentially by the data store and the combined results returned.

```
{
  "compareValue": COMPARE_VALUE, /* optional */
  "higherValuesAreBetter": BOOLEAN, /* optional */
  "chartType": STRING, /* optional */
  "queries":[
    {
      "source": STRING, /* the name of the data source to query */
      "maxresults": NUMBER, /* optional */
      "offset": NUMBER, /* optional */
      "starttime": NUMBER, /* optional */ /* ms since the epoch UTC */
      "endtime": NUMBER, /* optional */
      "constraints":[
        {
          "fieldname": STRING,
          "type": CONSTRAINT_TYPE,
          "value": CONSTRAINT_VALUE, /* optional */
        },
        ...
      ]
    }
  ]
}
```

```
COMPARE_VALUE = STRING|BOOLEAN|NUMBER
CONSTRAINT_VALUE = STRING|BOOLEAN|NUMBER|OBJECT*
CONSTRAINT_TYPE =
'<' | '<=' | '>' | '>=' | '==' | '!=' | 'REGEX' | 'SUM' | 'COUNT' | 'AVERAGE' | 'MIN' | 'MAX' | 'GROUP' | 'EXCLUDE' | 'QUANTILES' | 'SORT' | 'VALUE'
```

- * For constraint values, when an object with one field 'paramId' is present this field is substituted (by the StorageService) with a parameter passed to the query before the query is sent to a data store. This is an internal feature and the data store implementation does not need to handle objects as constraints.

Constraint types

Type	Req. Field name	Req. Value	Description
<, >, <=, >=, ==, !=	Yes	Yes	A constraint, the value of the field must pass this condition to be included
REGEX	Yes	Yes	A regular expression to match the value against. The format depends on the Data Store used. MongoDB supports PCRE format (http://docs.mongodb.org/manual/reference/operator/query/regex/#op. S regex). The KMP 'may' support SQL style LIKE syntax or regexp but it's not known at this time.
SUM	Yes	No	Sum all of the values in the given field, applies to numbers only.
COUNT	No	No	Count the number of records returned by the query
AVERAGE	Yes	No	Return the average of the values in the field specified, applies to numbers only
MIN	Yes	No	Return the minimum value in the field specified, applies to numbers only
MAX	Yes	No	Return the maximum value in the field specified, applies to numbers only
GROUP	Yes	No	Group the results by the given field name
EXCLUDE	Yes	No	Exclude the field name from the results
QUANTILES	No	Yes	Calculate the quantiles for the query results. The value is interpreted as the quantiles to get, e.g. "0.0,0.25,0.5,0.75,1.0" will retrieve five values representing four quartiles. To calculate quantiles a sort order constraint must be specified as well otherwise the results are meaningless.
SORT	Yes	Yes	Sort the results based on the values in the given field. If value >= 0 sort ascending, otherwise descending.
VALUE	No	Yes	Pass through the value to the result of the query.

Browser SVG Support

Source: <http://caniuse.com/svg>

= Supported
 = Not supported
 = Partially supported
 = Support unknown

SVG (basic support) - Recommendation

Method of displaying basic Vector Graphics features using the embed or object elements

Resources: [SVG Web: Flash-based polyfill](#) [Wikipedia](#) [Web-based SVG editor](#) [SVG showcase site](#)
[A List Apart article](#) [has.js test](#)

Global user stats *:

Support:	83.56%
Partial support:	0.02%
Total:	83.58%

	IE	Firefox	Chrome	Safari	Opera	iOS Safari	Opera Mini	Android Browser	Blackberry Browser	Opera Mobile	Chrome for Android	Firefox for Android	IE Mobile
26 versions back			4.0										
25 versions back			5.0										
24 versions back		2.0	6.0										
23 versions back		3.0	7.0										
22 versions back		3.5	8.0										
21 versions back		3.6	9.0										
20 versions back		4.0	10.0										
19 versions back		5.0	11.0										
18 versions back		6.0	12.0										
17 versions back		7.0	13.0										
16 versions back		8.0	14.0										
15 versions back		9.0	15.0										
14 versions back		10.0	16.0										
13 versions back		11.0	17.0		9.0								
12 versions back		12.0	18.0		9.5-9.6								
11 versions back		13.0	19.0		10.0-10.1								
10 versions back		14.0	20.0		10.5								
9 versions back		15.0	21.0		10.6								
8 versions back		16.0	22.0		11.0								
7 versions back		17.0	23.0		11.1								
6 versions back	5.5	18.0	24.0		11.5			2.1		10.0			
5 versions back	6.0	19.0	25.0	3.1	11.6	3.2		2.2		11.0			
4 versions back	7.0	20.0	26.0	3.2	12.0	4.0-4.1		2.3		11.1			
3 versions back	8.0	21.0	27.0	4.0	12.1	4.2-4.3		3.0		11.5			
2 versions back	9.0	22.0	28.0	5.0	15.0	5.0-5.1		4.0		12.0			
Previous version	10.0	23.0	29.0	5.1	16.0	6.0-6.1		4.1	7.0	12.1			
Current	11.0	24.0	30.0	6.0	17.0	7.0	5.0-7.0	4.2-4.3	10.0	16.0	29.0	24.0	10.0
Near future		25.0	31.0	7.0	18.0								
Farther future		26.0	32.0										

Browser JSON Parsing Support

Show options

= Supported
 = Not supported
 = Partially supported
 = Support unknown

JSON parsing - other

Method of converting JavaScript objects to JSON strings and JSON back to objects using `JSON.stringify()` and `JSON.parse()`

Usage stats:

Global

Support:

93.33%

Show all versions	IE	Firefox	Chrome	Safari	Opera	IOS Safari	Opera Mini	Android Browser	Blackberry Browser	IE Mobile
						3.2		2.1		
						4.0-4.1		2.2		
						4.2-4.3		2.3		
	8.0					4.2-4.3		3.0		
	9.0		28.0	5.1		5.0-5.1		4.0		
	10.0	23.0	29.0	6.0		6.0-6.1		4.1	7.0	
Current	11.0	24.0	30.0	7.0	17.0	7.0	5.0-7.0	4.2-4.3	10.0	10.0
Near future		25.0	31.0		18.0					

Notes

Known issues (0)

Resources (5)

Feedback

Edit on GitHub

Requires document to be in IE8+ standards mode to work in IE8.

Source: <http://caniuse.com/#search=json>

Full project file list

Generated class files have been omitted.

```

├── application.properties
├── grails-app
│   ├── conf
│   │   ├── ApplicationResources.groovy
│   │   ├── BootStrap.groovy
│   │   ├── BuildConfig.groovy
│   │   ├── Config.groovy
│   │   ├── DataSource.groovy
│   │   ├── UrlMappings.groovy
│   │   ├── WebXmlConfig.groovy
│   │   ├── ehcache.xml
│   │   ├── hibernate
│   │   └── spring
│   │       └── resources.groovy
│   ├── controllers
│   │   ├── LoginController.groovy
│   │   ├── LogoutController.groovy
│   │   └── org
│   │       └── learnovatecentre
│   │           └── evade
│   │               ├── AdminController.groovy
│   │               ├── AnalyticsController.groovy
│   │               ├── ApiController.groovy
│   │               ├── CaptureController.groovy
│   │               └── VisualisationController.groovy
│   └── domain

```



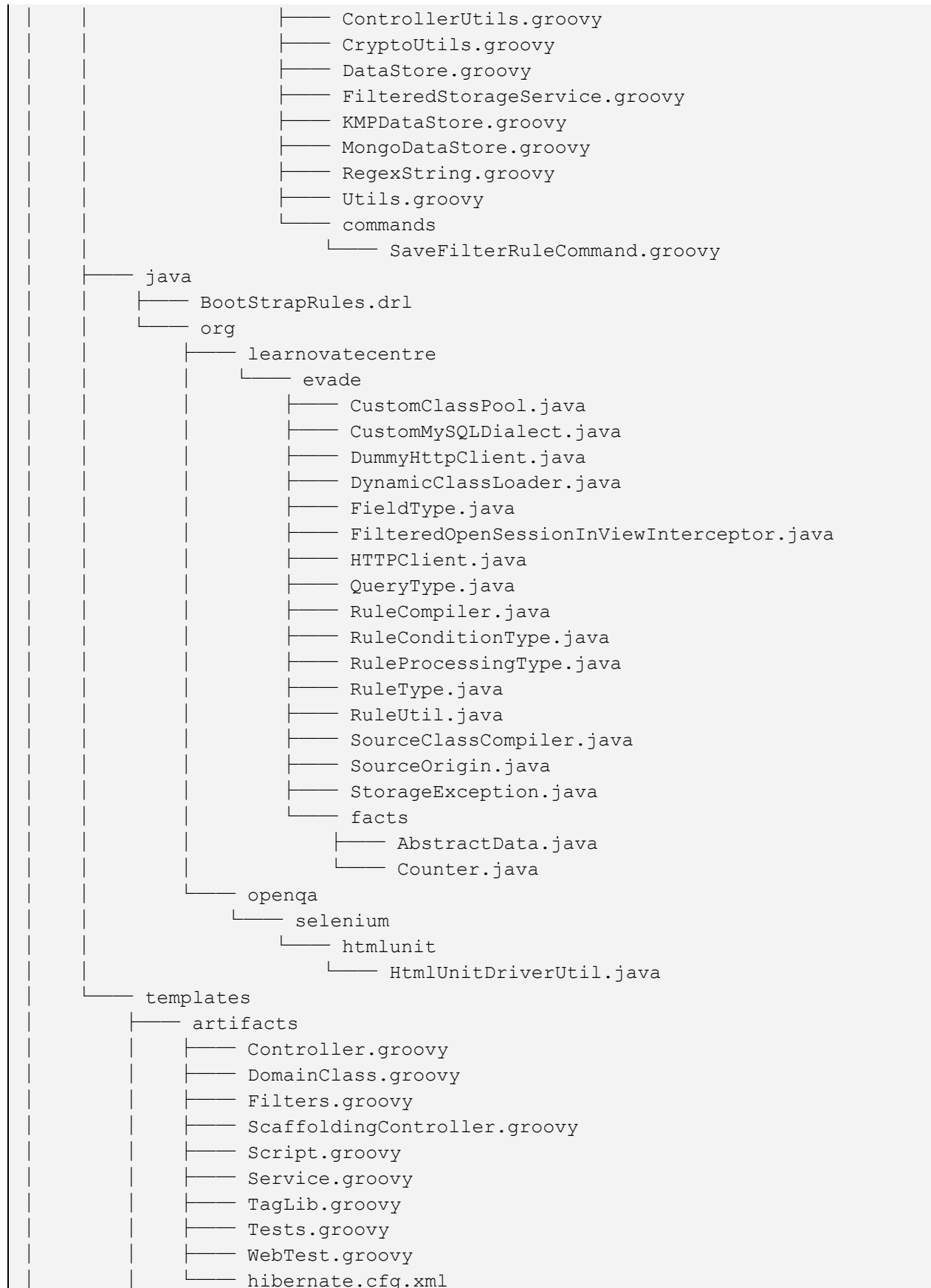
```

taglib
├── evade
│   └── UtilsTagLib.groovy
├── utils
└── views
    ├── admin
    │   ├── _editGroup.gsp
    │   ├── _editUser.gsp
    │   ├── datastore.gsp
    │   ├── index.gsp
    │   ├── logging.gsp
    │   ├── showGroup.gsp
    │   └── showUser.gsp
    ├── analytics
    │   ├── _ajaxConditionForm.gsp
    │   ├── _ajaxProcessingForm.gsp
    │   ├── _configBlockFILTER.gsp
    │   ├── _configBlockTHRESHOLD.gsp
    │   ├── _editQuery.gsp
    │   ├── _fieldList.gsp
    │   ├── _fieldRuleList.gsp
    │   ├── _inputSourceList.gsp
    │   ├── _ruleList.gsp
    │   ├── _sourceList.gsp
    │   ├── analyse.gsp
    │   ├── create.gsp
    │   ├── createQuery.gsp
    │   ├── editQuery.gsp
    │   ├── index.gsp
    │   └── show.gsp
    ├── api
    │   ├── index.gsp
    │   ├── usageAdapterHTML.gsp
    │   └── usageHTML.gsp
    └── capture
        ├── _adapterList.gsp
        ├── _editAdapter.gsp
        ├── _editField.gsp
        ├── _editMapping.gsp
        ├── _editSource.gsp
        ├── _fieldList.gsp
        ├── _fieldListEditable.gsp
        ├── _mappingList.gsp
        ├── _navigation.gsp
        ├── _sourceList.gsp
        ├── _sourceSummaryList.gsp
        ├── create.gsp
        ├── createAdapter.gsp
        ├── edit.gsp
        └── editAdapter.gsp
    
```

```

├── index.gsp
├── list.gsp
├── show.gsp
├── showAdapter.gsp
├── usage.gsp
├── error.gsp
├── index.gsp
├── layouts
│   ├── main.gsp
│   ├── new_main.gsp
│   └── old_main.gsp
├── login
│   ├── auth.gsp
│   └── denied.gsp
├── old_index.gsp
├── rule
│   ├── _filter.gsp
│   ├── _generic.gsp
│   ├── _ruleHeader.gsp
│   ├── _ruleStart.gsp
│   └── _sum.gsp
├── shared
│   ├── _flashError.gsp
│   ├── _left-nav.gsp
│   └── _navbar.gsp
├── visualisation
│   ├── _editEmbed.gsp
│   ├── _embed.gsp
│   ├── _ruleQueryList.gsp
│   ├── _showEmbed.gsp
│   ├── _showQuery.gsp
│   ├── _showRuleQuery.gsp
│   ├── _sourceFieldList.gsp
│   ├── _sourceList.gsp
│   ├── _visualisationForm.gsp
│   ├── demo.gsp
│   ├── edit.gsp
│   ├── index.gsp
│   └── stream2.gsp
├── lib
│   └── mariadb-java-client-1.1.5.jar
├── readme.txt
├── scripts
├── src
│   ├── groovy
│   │   └── org
│   │       └── learnovatecentre
│   │           └── evade
│   │               ├── AjaxStream.groovy
│   │               └── ApiTarget.groovy

```



```

├── scaffolding
│   ├── Controller.groovy
│   ├── Test.groovy
│   ├── _form.gsp
│   ├── create.gsp
│   ├── edit.gsp
│   ├── list.gsp
│   ├── renderEditor.template
│   └── show.gsp
├── testing
│   ├── Controller.groovy
│   ├── DomainClass.groovy
│   ├── Filters.groovy
│   ├── Generic.groovy
│   ├── Service.groovy
│   ├── TagLib.groovy
│   └── UnitTest.groovy
└── war
    └── web.xml
stacktrace.log
target
├── classes
├── dependency-report
├── reports
├── stacktrace.log
├── test-classes
├── test-reports
├── tomcat-err.txt
└── tomcat-out.txt
target-eclipse
test
web-app
├── META-INF
├── WEB-INF
├── css
│   ├── bootstrap2-responsive.css
│   ├── bootstrap2.css
│   ├── bootstrap3-theme.min.css
│   ├── bootstrap3.min.css
│   ├── errors.css
│   ├── font.css
│   ├── main.css
│   ├── mobile.css
│   └── old_main.css
└── images
    ├── apple-touch-icon-retina.png
    ├── apple-touch-icon.png
    ├── favicon.ico
    ├── glyphicons-halflings-white.png
    └── glyphicons-halflings.png
  
```

```
├─── rails_logo.jpg
├─── rails_logo.png
├─── leftnav_btm.png
├─── leftnav_midstretch.png
├─── leftnav_top.png
├─── loading.gif
├─── skin
│   ├─── database_add.png
│   ├─── database_delete.png
│   ├─── database_edit.png
│   ├─── database_save.png
│   ├─── database_table.png
│   ├─── exclamation.png
│   ├─── house.png
│   ├─── information.png
│   ├─── shadow.jpg
│   ├─── sorted_asc.gif
│   └─── sorted_desc.gif
├─── spinner.gif
├─── springsource.png
└─── js
    ├─── application.js
    ├─── bootstrap2.min.js
    ├─── bootstrap3.min.js
    ├─── d3.v3.min.js
    ├─── jquery.stream-1.2-mod.js
    ├─── jquery.stream-1.2.min.js
    ├─── query_builder.js
    ├─── rule.js
    └─── visualisations.js
```