Vicarious Learning in Informatics
YouTute: Interface Design

Neil Mayo, nmayo@inf.ed.ac.uk
Informatics, University of Edinburgh

May 26, 2008

Contents

List of Figures

1 Introduction

This document describes the design and implementation of the YouTute web interface, from the visual interface to the client-side dynamics and the server-side database management. There is also some discussion of the steps required to setup the system.

The system uses the following technologies: **RED5** open source streaming server for flash videos; **MySQL** for the database backend; **Java Servlets and JSP** served via **Tomcat**, allowing database access; and **JavaScript** for client-side reactivity.

Several libraries have also been used, including: **MySQL Connector/J** for connecting Java to MySQL; **Apache Commons** supplies various utility functions; **JSTL** and **Core** tag libraries provide common JSP functionality; **YUI** The Yahoo! User Interface Library, for JavaScript interface support; **JW FLV Media Player**, an open source flash media player.

The system is also broadly based upon the AJAX umbrella of technologies, as we primarily use asynchronous HTTP requests to initiate communication with the server from JavaScript and perform page updates. However XML is not used.

1.1 Motivation

YouTute was started as an alternative to the *Diver* system at Stanford, which we found did not quite meet our needs, in particular due to the compression applied to submitted videos. We require the ability to play three videos simultaneously, a reasonable resolution for the smartboard videos, and also high-quality sound.

> There is a YouTute wiki at `https://wiki.inf.ed.ac.uk/YouTute`.
>
> The SVN repository for DICE users is at
> `svn+ssh://username@svn.inf.ed.ac.uk/svn/youtute`

2 System Overview

YouTute is designed to allow students to select pre-recorded tutorial videos and watch them to aid revision. In addition to basic playback it is possible to designate subsections of the videos to focus on a question of interest. We shall use the term *Tute* for our video sections; the equivalent of Diver's *Dives*.

There should also be functionality allowing the Tutes to be tagged with short descriptive tags which can then be used in providing search facilities and resource sharing. Longer comments can be applied to specific points in the videos to share thoughts with others or as an aide-memoire. Tutorial materials such as PDFs are provided for reference.

A major goal in the interface was to avoid reloading the page as far as possible, in order both to avoid the repeated overhead of building and initialising the page and its (heavyweight) elements, and to allow the players to continue while updates are being performed, hence improving the sense of workflow. This suggested the use of AJAX and asynchronous HTTP requests.

2.1 Server Backend

There are two servers; one streams the video using RED5 and is based in the architecture department; the other is a Tomcat server running on `fordyce` in Informatics, and is tied to the frontend. This is the one described in detail here.

2.1.1 Database

The basic information supporting the interface is maintained in a database on the server (see DB-design.pdf for a discussion of the design of this database and the basic schema). User-created information such as tutes, tags and comments are appended to this database.

Interaction with the database is achieved through two standard approaches; a) the use of JSTL tags in the JSP pages, and b) interaction with Java servlets provided on the server.

All server-side Java is encapsulated in the package `uk.ac.ed.inf.youtute` – this includes the subpackages `user` for tracking user info via a bean; `database` for providing various types of database access; and `logging` to handle both server logging and the logging of client-side events through JavaScript. Some of the most important classes are summarised below:

database.ConnectionFactory sets up a connection to the database.

database.QueryEngine abstracts the usage of the database connection, providing an interface for executing queries and updates, and retrieving results.

logging.ServerLogger provides a logger and methods for server-side logging.

2.1.2 Servlets

There are several servlets available to support the interface (note that these are mapped to specific contextual paths in Tomcat):

database.QueryRequest runs a query and returns the results in a HTTP response, in customised formats and properly escaped. Mapped to `/youtute/query`.

logging.ClientLogger saves log requests forwarded from a JavaScript logger running in the client. Mapped to `/youtute/logger`.

There are also several Java beans that provide access to required information:

DataQueryBean provides access to the results of common pre-defined queries, and also runs some update queries.

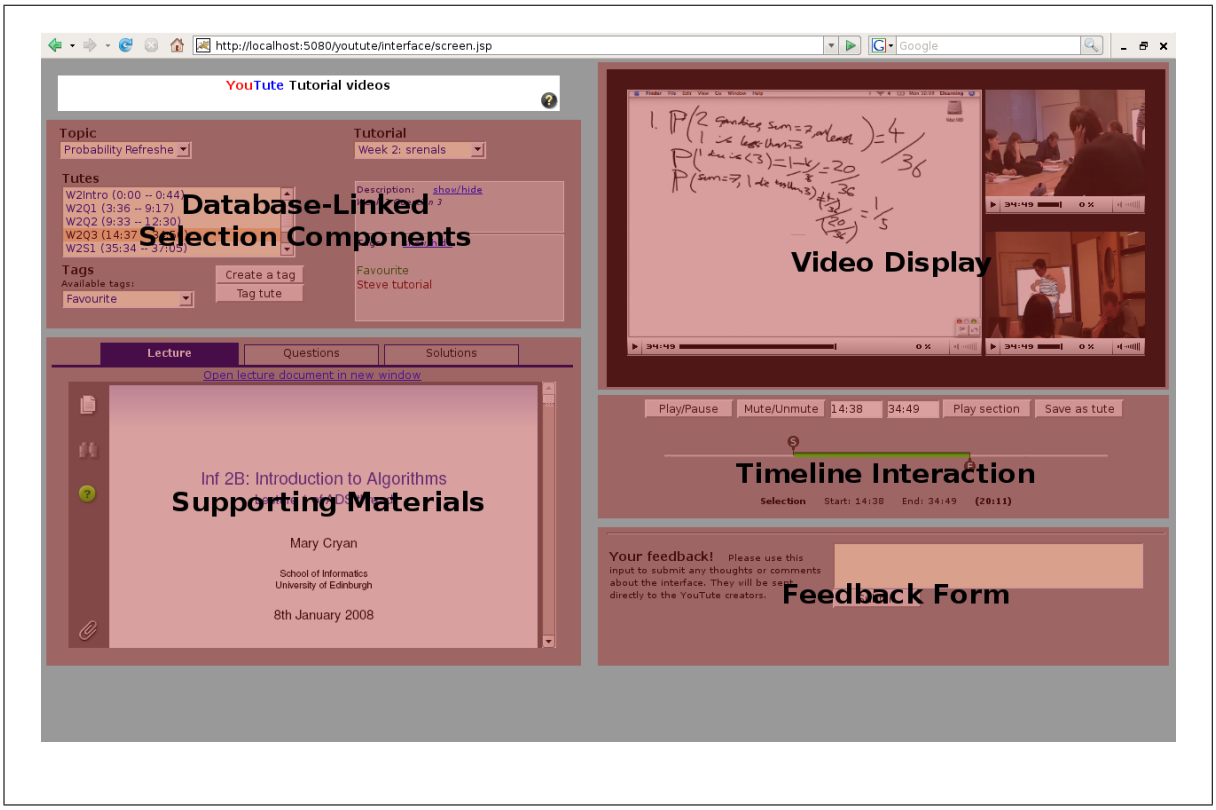
UserBean is a repository of information for a logged-in user; records basic info such as id, name, email; and context-specific information extracted from the database in advance, such as a user's tutes, comments, tags.

All these classes are self-documenting, and appropriate Javadocs can be generated from the code using javadoc.

2.2 Client User Interface

The front-end to the interface consists of a web-based GUI based on Java Server Pages, and using JavaScript to mediate interaction of components with one another and with the server. All the JavaScript classes are self-documenting; they are annotated using the Javadoc standard, and appropriate 'Javadocs' can be generated from the code using JSDoc (`http://jsdoc.sourceforge.net/`).

Figure 1: The user interface, showing the major regions of activity.



The primary client interface consists of several regions of activity as indicated in 1. The selection components are linked to the database and allow the user to narrow down the area of interest for revision. Once a tutorial is selected, appropriate materials are loaded and displayed, and the tutorial's videos are loaded into the video display area. At this point a timeline is also displayed which represents the progress of the videos and the extent of the tute selection.

3 Authentication

Before providing the interface to the user, it is necessary to authenticate them via a login page. Only users listed in the database `user` table are granted access to the system.

Authentication is managed using JSTL within JSP; each protected page starts with some JSTL code which first checks whether a valid `user.UserBean` has been created, and if not redirects to the login page. The level of authentication can be used to restrict access to certain resources, and to authorise tag and tute edits, comments and other user-modifiable information; though this is not currently used.

A `UserBean` created for a valid login is used to maintain user-specific records such as name, email, and user-created tags and tutes, and may ultimately be used to track other user-specific information such as user interface preferences.

4 Database-Linked Selection Components

Before using the video-editing interface, the user must narrow down the selection using the combo boxes and lists provided. Once a tutorial is selected, the videos and materials can be loaded, as can a list of tutes. Because of the scale of the update, this is done by reloading the page by submitting the page's central form. Other components initiate asynchronous updates using AJAX.

Figure 2: Database-Linked Selection Components and their effects when a selection is made.

Component	**Effect**
Topic	Updates the tutorial list.
Tutorial	Updates the tutes list and loads videos and materials for the selected tutorial.
Tutes	Updates the start and end thumbs in the slider.
Tags	No effect.

When an asynchronous update is required based on a new selection, the `/youtute/query` (`database.QueryRequest`) servlet performs database updates and provides updated query results. New results are decoded and written into the relevant components using JavaScript. A `DataQueryBean` is used to provide an interface to common queries, and also maintains values from the page's form. It is used when the page is reloaded on the selection of a tutorial.

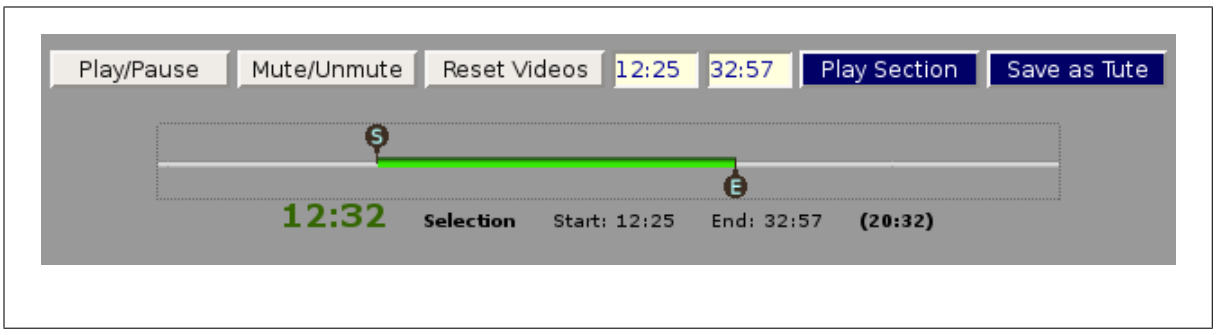
5 Video Playback, Control and Editing

The 'video display' section of the screen is central to YouTute and is supported by the 'timeline interaction' section which contains widgets for interaction with the videos, and display components indicating the status of the videos. The videos themselves are played in the Jeroen Wijering FLV Player, which provides a JavaScript API for controlling and monitoring the player.

5.1 The Timeline

Although there are three videos representing the tutorial, they should be treated as a single unit, playing. pausing, scrubbing and muting in parallel. We thus introduce the concept of a *Timeline* as an abstraction of the three videos, and use this to coordinate them with the slider; this functionality is contained within the `tuteSlider` class.

Figure 3: The timeline controls – showing video control buttons, the slider with draggable start and end thumbs, and time displays.



The slider represents the timeline, the playable range of video; and start and end thumbs on the slider allow a range to be described within the timeline. The associated displays indicate the status of this overall timeline. This includes an indication of progress through the timeline, and of the start, end and length of a tute selection. There are a number of buttons facilitating interaction with the timeline; it is played, paused and muted as a unit. Further buttons allow the current selection to be played or to be saved as a new tute. Methods updating the time display should accept the most accurate values available, which in this case is seconds, as received from the JW player.

The timeline is supported by several variables in `playerControl.js`, namely `timelinePlaying`, `timelineMuted` and `timelinePos` which represent the timeline's playing state, muted state and position respectively.

Note: the timeline may eventually be rescaled to represent the selected tute, or we may separate timeline progress and tute selection components.

Note: although this is not implemented currently, the Timeline approach also allows an offset to be specified for each video, so that videos of differing length can be coordinated to a common 'zero' time.

5.2 The synchPlayer class

Each video in the display is represented by a `synchPlayer`, an abstraction which encapsulates the streamed video and the player instance, and also adds peripheral functions and variables to track the synchronisation of the videos.

Each synchPlayer has a Boolean variable `playing` which is used in playing/pausing the timeline. In case video play states become inconsistent (e.g. if the user plays or pauses one manually by clicking on the player), the variable is checked and the play state is toggled only if required to match `timelinePlaying`. The `playing` states of the synchPlayers are updated when state updates are received from the JW player.

One of the synchPlayers (by default the smartboard video) acts as a 'master' player – this is the player which provides the sound (the others are muted), and which is consulted for the timeline length and current position. In fact videos vary in length, which can be overcome by using a timeline offset as described above.

5.3 Event callbacks

The JW player `getUpdate()` callback informs JavaScript of player events, providing the player id, the type of event, and any supporting parameters. These updates are used as follows:

state (stop/buffer/play) – make sure the video is doing what it should according to `timelinePlaying`, and update the video's `playing` state variable. Note that basic synchronisation could be implemented in a similar way.

time (elapsed, remaining) – update timeline values (position, display); try and set the timeline length; check whether to stop the playback becuase the end point has been reached (`tuteEnd`, represented by slider end thumb). This event is only acknowledged for the master player.

load (percentage loaded) – if the load percentage is 100, set the player's `fullyBuffered` variable and decrement the `numVideosStillLoading` counter in `playerControl.js`.

Timeline length The appearance of the timeline is delayed until the video length is known, so it can be scaled correctly. The video length cannot be directly requested from the player, but can only

be calculated from the update callbacks, so this introduces a delay during which the timeline slider is hidden. The videos must be started programmatically to force them to generate update events.

5.4 Synchronisation

Note: *Synchronisation is not currently implemented. A temporary fix if the videos become staggered is to click on the start thumb in the slider; this will reorient all the players to the position of the thumb.*

There are several possible approaches to video synchronisation, including a) attempting to sync videos locally in JavaScript; b) synchronising streams in Red5 using metadata?; c) synchronising using Flash, within the players?.

5.4.1 JavaScript Synchronisation

Currently this is probably the easiest way to perform video synchronisation for the system. Broadly, the approach is to monitor the synchronisation state by checking the positions of the videos at specified intervals using the JW player callbacks. If the gap between videos exceeds a given threshold, the video positions are realigned.

Note: the JW player load event provides the percentage loaded, but it can sometimes revert to a lower value when lots of buffering is occurring. This can cause problems as currently each player's `fullyBuffered` boolean is only set once.

There are three main issues relating to synchronisation, each of which may require a slightly different approach:

- synchronising when starting/playing
- staying in sync when videos are buffering
- synchronising when scrubbing/releasing

The approach to these issues is described in the following paragraphs.

When playing A `time` update is received every second from the JW player, providing the time elapsed and time remaining. Check the position of the master video every n updates and force other videos to align if they have strayed by greater than δ, e.g. 2 seconds. It is unclear how much these checks will affect performance, especially when video is being scrubbed using the timeline slider controls. The values for n and δ must be chosen carefully.

Note: The alignment must be done in relation to a master video or there will be a nasty update loop.

When buffering The approach described will not necessarily work while buffering ($state = 1$), so perhaps the players should wait or pause until all videos are fully loaded ($load = 100$), or buffering is not occurring.

When scrubbing During manual scrubbing the sync check should be disabled, but we should maintain the correct play/pause states and re-enable checking on the release of the slider. It may also be desirable to keep up with the position of the slider – this has already been implemented but disabled because it makes the slider unresponsive.

6 Interface Control with JavaScript

The interface interactivity is provided by several JavaScript modules, along with several CSS files which provide both visual settings and settings tied to the behaviour of divs and other page elements.

6.1 The JSP Interface

The main page is `screen.jsp`. Its structure is:

- JSTL Taglib and Java/JSP imports.
- Login check (this is duplicated in other JSPs to provide security).
- Bean which performs canned queries.
- Database to support JSTL SQL tags.
- Save a new tute via a database update if new tute information was submitted from the form.
- JavaScript imports and stylesheets for YouTute and third party libraries.
- Inclusion of hidden variables used to maintain supplementary information from the database.
- Form, containing the interface components.

Some sections of the main page `screen.jsp` are maintained in separate files for clarity, in the `components/` directory:

`hidden-vars.jsp` This contains hidden variables used to maintain supplementary information from the database. They are included outside of the main form, and so are not submitted. This could alternatively be achieved with a bean, however the information cached in this way is related to the current tutorial and so is expected to remain static while the page is not reloaded; the variables are updated to reflect a new tutorial selection or a new tute, both of which update the whole page.

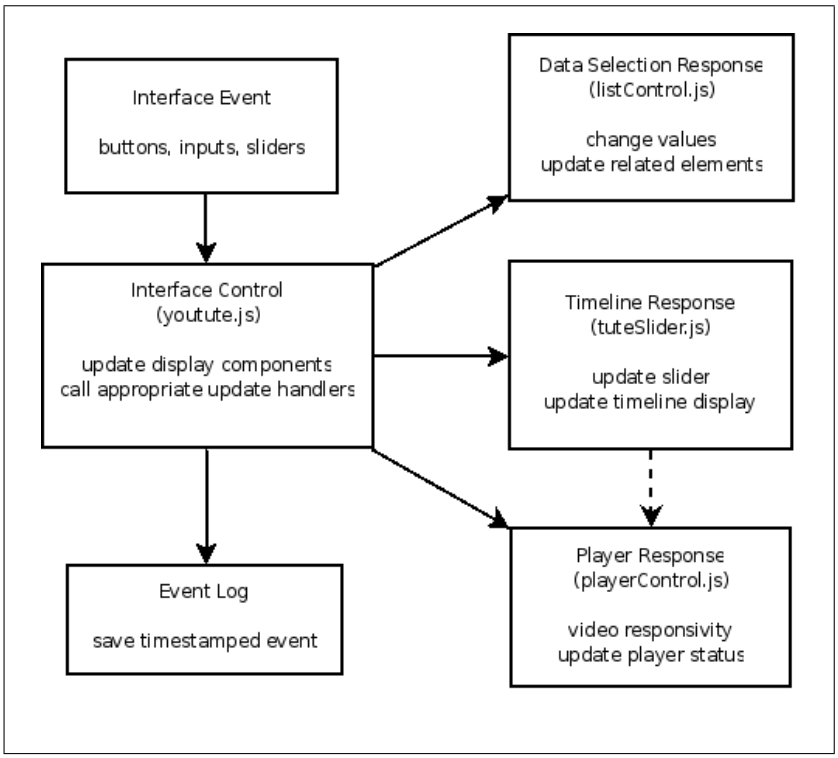
`videos-layout*.jsp` An abstraction of video layouts so they can easily be swapped, even through JavaScript.

`header-row.jsp` A basic navigational header used in the login and welcome pages.

6.2 Interface Events

The process of handling interface events is illustrated in figure 4. User-generated events are initially picked up by `youtute.js`, which performs immediate interface updates, logs the event, and then calls update handlers in other modules. `listControl.js` updates the lists and combos in the interface by requesting data updates from the server. `tuteSlider.js` updates the video controls and display, and `playerControl.js` ensures the players maintain the correct state and respond to the interface; it is sometimes also called directly by `tuteSlider.js` or `listControl.js`. The `asynchRequest.js` module is used by several modules to communicate asynchronously with the `/youtute/query` servlet on the server.

Figure 4: How interface interaction is handled in JavaScript.



6.3 Querying and Updating the Database Asynchronously

The module `asynch.js` uses YUI to make asynchronous requests to the `/youtute/query` servlet. Several types of "request" are available, depending on whether the results should be used to fill a combo, a text element, a hidden form field, or just sent to the server and forgotten (as in the case of client logs). The types of request serviced are defined in a set of constants in the servlet class.

Behind the scenes, the `QueryRequest` servlet runs a query and returns the results in an HTTP response. The query may be a SELECT call or an UPDATE, and the result returned to the client may contain query results, an empty string, or the ID of the newly created entry (not implemented).

The request data should be of the form "topic=2?tutorial=1" and should not contain the restricted characters & % < > #. Checks are performed in JavaScript for these characters in user input. Results returned from the servlet are HTML-escaped and translated into a custom string format. This means the result strings can be placed straight into an HTML page once decoded.

Because of the amount of data involved in saving a new tute, this is done through submitting the form and reloading the page, with the properties of the new tute recorded as hidden fields.

7 Logging

Logging is performed both in the server (Java) and in the client (JavaScript). The client logs are forwarded to a servlet for processing.

7.1 Client logging

Logging of events in the client is achieved using a JavaScript logger which posts log messages to the server asynchronously. YouTute uses the `log4javascript` package, the logger initialised with a `log4javascript.AjaxAppender`.

The `/youtute/logger` (ClientLogger) servlet receives requests from the JavaScript logger running in the client, formats them and passes them on to the `ServerLogger`.

Log messages are formatted according to the scheme in figure 5.

Figure 5: Client log message format.

```
[loglevelTIMESTAMPuser]  message
        [INFO1210161374neil]  neil logged in
        [INFO1211536644neil]  page initialised
        [INFO1211536644neil]  play timeline
        [INFO1211536646neil]  show help
        [INFO1211548411neil]  topic inf2b_tut3
```

Events are logged for as much of the user interaction as possible. The messages generated in the client are kept simple and consistent so the log files may be more easily parsed. Figure 6 shows a list of the events logged during interaction with the interface. User logins are also logged.

Note: we cannot log ids for newly-created records unless we get them back from the request, and log the event at the end of the event handlers. This is not yet configured in the server.

7.2 Server logging

Server side logging is performed in `ServerLogger` using the `org.apache.log4j` package. Currently there are no logging calls in the server code; only the forwarded client logs are recorded.

To Do: *Log user and browser; add a session id to the user bean and use it in logging*

Note: it is possible to pass a callback servlet to the JW player for logging video play/stop statistics. This is not used currently.

8 Usability Discussion

There are several unresolved questions concerning the presentation of information in the interface:

- is the tutorial/tute distinction useful or confusing? should it be collapsed?
- does the layout encourage a sensible workflow? is it clear and does the order of selection make sense?
- is it useful to create custom tutes in addition to the predefined question tutes? are other people's tutes useful?
- are tags useful? would a tag cloud be a useful visualisation?
- should there be a many-to-many relationship between topics and tutorials? (see database document)

Figure 6: A list of logged interface events for each region of the screen.

- Selection
  - topic select ID
  - tutorial select ID
  - tute select ID
  - tag select ID/NAME
  - materials select X [lecture(n), questions, solutions]
- Buttons
  - create tag (ID)/NAME
  - tag tute X with tag (ID)/NAME
  - reset videos
  - play/pause
  - mute/unmute
  - help
  - play section S, E
  - save as tute (ID), S, E
  - submit feedback (ID)
- Slider
  - start drag to X
  - end drag to X
  - start input enter T
  - end input enter T

Note: We hope to answer some of these questions during a student focus group.

8.1 Clarity of the Tutorial/Tute Distinction

The distinction between tutorials and tutes is a bit fuzzy, particularly because an entire tutorial video is loaded in order to show a tute; practically a tute is only a range within a tutorial rather than a separate entity. Functionally a tutorial is just a tute spanning the full-range of the video.

Should tutorial become a subtype of tute? Note the common relations and the similar queries for each type. Both tutes and tutorials should be taggable, can be loaded and played, searched and so on. The important thing to consider here is whether we need to distinguish between tute/tutorials at all, and if so how (particularly within the interface page). Further questions are whether all tutes are direct children of tutorials, or is it a recursive structure; and should the timeline range be rescaled when playing a tute?

8.2 Alternative layout schemes

To Do: *include rough diagrams of alternative layouts*

8.3 Alternative interaction

An important question is how the students will interact with the interface, and how to most naturally represent these potential work flows. Tasks include video editing, video watching, tute creation, browsing of existing tutes and others' tutes, search. There is some justification for separating the tute edit process from the revision process, and thinking of the interaction as an ordered sequence of steps – make a set of tutes, and then view, tag and join them in a separate environment.

What is the revision procedure? An example procedure might be to identify some tutes, create them, then choose some associated tutes via 'search' or 'related' links, and view them all on a page together (YouTube style with thumbnails and short description, prominent tags, links to other videos).

Is the selection process intuitive/helpful – drilling down onto a set of videos/tutes by selecting a topic, a tutorial, then a tute? Should the page representation change? (e.g. choose topic/tutorial in a different page and then provide a page dedicated to all the tutes for that particular topic/tutorial).

9 Future Work

Note: Tasks and fixes for the future can be submitted and discussed at the YouTute wiki `https://wiki.inf.ed.ac.uk/YouTute`.

Some of the major tasks are described here:

- Build the webapp to a `war` file.
- Database – connection pooling and thread safety, prepared queries, restructure the backend.
- Personalise queries to be user-specific.
- Search interface – search via tags, topics, questions?
- Related clips – same tutor, date, topic, keywords/tags
- Comments
- Allow multiple lecture materials.
- Ability to play a sequence of tutes.
- Pointer links between tutes e.g. A explains B ?
- Provide a web-embeddable link to a tute sequence
- Record modification time for tutes, comments, tags
- Private server backend for processing and viewing logs and database stats. (views of log file per user, new tags/tutes/comments per user, feedback) [HTML, Perl, XSL]
- Go to specific pdf page – split pdfs into individual pages or record a question's page in the database.
- Provide other materials – transcript, audio mp3
- Record personal info in user bean such as tutes, tags/taggings
- Allow tag/tute deletion/modification
- Only show unused tags for a selected tute.

- Distinguish between universal and personal taggings
- Session Management – add password change and logout screens; remember login via cookie; use EASE authentication
- Error handling

User Interface ideas:

- Tag Cloud with links to search result showing tagged tutes
- Enlarge (pdf) material via enlarge image to the right of the tabs
- Add overlib popups showing help information for each widget
- Get feedback via a modal window like help (reduces screen clutter).
- Put materials tabs down the side
- Record user preferences in bean (mute, screen/video layout, show tags/desc, remember login, show intro/search/edit on login, remember last tutorial/topic, all/personal tag display)
- Allow (and display) transitive taggings via tutorial, topic
- Show all tutorials by default; if selected, update the selected topic

APPENDIX

A Tomcat Server Configuration

Several changes must be made to Tomcat configuration files in order to support YouTute. This includes the system-wide files `server.xml`, `tomcat-users.xml` and `web.xml` in `$CATALINA_HOME/conf/`, and the `web.xml` for the `youtute` webapp itself.

A.1 Setting up Tomcat

A.1.1 conf/server.xml

A database resource must be added to the context of the YouTute webapp as follows. Note that the username and password for connecting to MySQL are supplied. MySQL also must be running on localhost. This resource can then be used by pages in the webapp.

```
<Context path="/youtute" docBase="youtute"
         debug="5" reloadable="true" crossContext="true">

  <Resource name="jdbc/YouTuteDB"
            auth="Container"
            type="javax.sql.DataSource"
            maxActive="100" maxIdle="30" maxWait="10000"
            username="root" password="youroot" driverClassName="com.mysql.jdbc.Driver"
            url="jdbc:mysql://localhost/youtute?autoReconnect=true"/>

</Context>
```

Basic authentication It is also possible to use the `conf/tomcat-users.xml` file for general site authentication; see settings for `GlobalNamingResources` in `server.xml`, and add users to `tomcat-users.xml`.

```
  <user username="neil" password="neil" roles="tomcat,admin,manager,ytuser"/>
```

Database authentication Authentication can be provided by reference to a table in the database, by setting up a security realm similar to this:

```
   <Realm className="org.apache.catalina.realm.JDBCRealm"
          driverName="org.gjt.mm.mysql.Driver"
          connectionURL="jdbc:mysql://localhost/youtute"
          connectionName="root" connectionPassword="youroot"
          userTable="user" userNameCol="user_id" userCredCol="password"
          userRoleTable="user_id" roleNameCol="role_name" />
```

A.2 Setting up YouTute

A.2.1 WEB-INF/web.xml

The file `<youtute>/WEB-INF/web.xml` is distributed with the code. It contains several customisations for YouTute, including declarations for taglibs, servlets and server resources.

Taglibs Each JSTL taglib must be set up with declarations – YouTute uses the core, sql and function libraries.

```
<taglib>
  <taglib-uri> http://java.sun.com/jsp/jstl/core </taglib-uri>
  <taglib-location>/WEB-INF/tlds/c.tld</taglib-location>
</taglib>
<taglib>
  <taglib-uri> http://java.sun.com/jsp/jstl/sql </taglib-uri>
  <taglib-location>/WEB-INF/tlds/sql.tld</taglib-location>
</taglib>
<taglib>
  <taglib-uri> http://java.sun.com/jsp/jstl/functions </taglib-uri>
  <taglib-location>/WEB-INF/tlds/fn.tld</taglib-location>
</taglib>
```

Servlets Each servlet must be declared and mapped to a relative address. We use a query servlet and a logger servlet:

```
<!-- QueryRequest servlet -->
<servlet>
  <servlet-name>query</servlet-name>
  <servlet-class>uk.ac.ed.inf.youtute.database.QueryRequest</servlet-class>
</servlet>

<!-- Client Logging servlet -->
<servlet>
  <servlet-name>logger</servlet-name>
  <servlet-class>uk.ac.ed.inf.youtute.logging.ClientLogger</servlet-class>
</servlet>

<!-- QueryRequest servlet mapping -->
<servlet-mapping>
  <servlet-name>query</servlet-name>
  <url-pattern>/query</url-pattern>
</servlet-mapping>

<!-- ClientLogger servlet mapping -->
<servlet-mapping>
  <servlet-name>logger</servlet-name>
  <url-pattern>/logger</url-pattern>
</servlet-mapping>
```

Resources A reference is declared to the database resource declared in `$CATALINA_HOME/conf/server.xml`:

```
  <resource-ref>
    <description>DB Connection</description>
    <res-ref-name>jdbc/YouTuteDB</res-ref-name>
    <res-type>javax.sql.DataSource</res-type>
    <res-auth>Container</res-auth>
  </resource-ref>
```

B External Supporting Libraries

YouTute makes use of a number of external libraries:

Name	Description	Deployment	Main use in YouTute
JSTL	Tag libs (jstl, standard).	Place `jars` in the Tomcat `common/lib/` directory. Place `tld` files for specific libraries in the YouTute WEB-INF directory.	Tags used in database access and control flow in setting up JSPs.
	`http://java.sun.com/products/jsp/jstl/`		
Apache Commons (dbcp, el, pool, collections, lang)	Java components and extensions.	Place in the Tomcat `common/lib/` directory.	DB Connection pooling (dbcp, pool), JSP Expression Language interpreter (el), Java Collection extensions (collections), extra Java lang functionality (lang).
	`http://commons.apache.org/`		
MySQL Connector/J	MySQL interface for Java JDBC.	Place in the Tomcat `common/lib/` directory.	Connecting the backend to the database.
	`http://www.mysql.com/products/connector/j/`		
JW Media Player	An FLV player with a JavaScript API.	Place `mediaplayer.swf` in the `youtute/interface` web directory.	Playing videos and controlling them via JavaScript.
	`http://www.jeroenwijering.com/?item=JW_FLV_Media_Player`		
YUI	Yahoo! JavaScript User Interface Components.	Place the `yui` directory in the Tomcat `webapps/` directory, alongside the `youtute` webapps directory.	TabView and Slider components in the interface. Asynchronous HTTP requests.
	`http://developer.yahoo.com/yui/`		
log4javascript	A JavaScript logger based on the Apache `log4j` Java logger.	Place in the web directory.	Logging of client activity.
	`http://sourceforge.net/projects/log4javascript`		
Unused Libraries			
overLIB	Tooltip functionality for JavaScript interfaces.	Place in the web directory.	
	`http://www.bosrup.com/web/overlib/`		
pdftk	Command-line PDF toolkit, including a page splitter.		
	`http://www.accesspdf.com/pdftk/`		
jsmin	Command-line JavaScript minimiser and obfuscator.	Run on JavaScript files to create compressed versions.	Increase efficiency?
	`http://www.crockford.com/javascript/jsmin.html`		
HTML::TagCloud	HTML tag cloud generator (Perl module).	Available as `libhtml-tagcloud-perl` in Ubuntu.	Might be useful to create basic tag clouds on the fly from tag data.