CLEMSON UNIVERSITY

MeTube

mmlab.cs.clemosn.edu/spring14/g8/Metube/web

Team G8

Yaolin Zhang
Yan Bai
Yunqing Zhang

04/29/2014

Table of Contents

1.Introduction

1.1 Project Overview

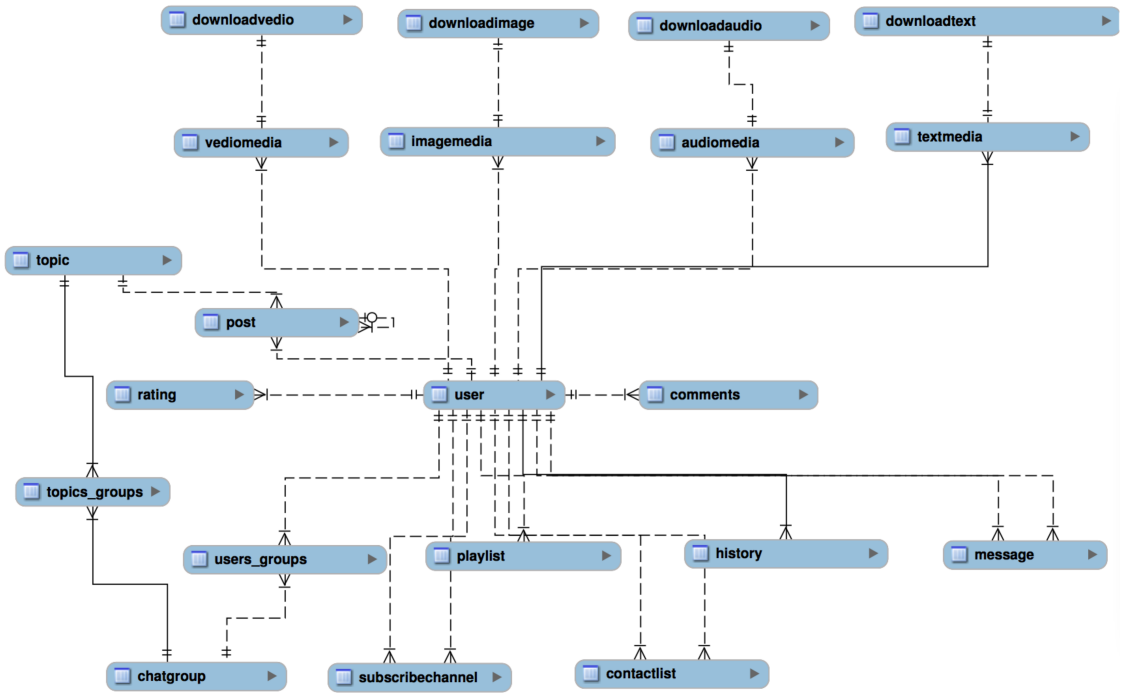
Along with more and more people surf the Internet and watch videos on website. Regularly, people want to share their favorite video with their friends. Increasing requirement, it is necessary to develop an online multimedia database system. Youtube is one of the most popular video-sharing website all over the world,on which users can upload, view and share videos. The project we dois similar with Youtube. It is worth mentioning that Hulu is one of the strongest competitor with Youtube .Hulu is a website and over-the-top (OTT) subscription service offering ad-supported on-demand streaming video of TV shows, movies, webisodes and other new media, trailers, clips, and behind-the-scenes footage from NBC, Fox, ABC etc.

The similarfunction with Youtube, better layout of webpage than Youtube. In our project, we try to implement the similar function with Youtube, imitate the webpage layout of Hulu.

1.2 Development Tool:

Based on *Linux Ubuntu*operating system, all the coding work is conducted. The server platform we use is *netbeans*. It is an integrated development environment (IDE) for developing primarily with Java, but also with other languages, in particular PHP, C/C++, and HTML5.To finish the project, some Server techniques are used like *Symphony*, *doctrine*,*PHP DQL*(sql),*Twig*(server view pages).Symfony is a PHP web application framework for MVC applications. Doctrine is a codification of beliefs or a body of teachings or instructions, taught principles or positions, as the body of teachings in a branch of knowledge or belief system.One of Doctrine's key features is the option to write database queries in a proprietary object oriented SQL dialect called Doctrine Query Language (DQL). Twig is a modern template engine for PHP Twig compiles templates down to plain optimized PHP code. The overheadcompared to regular PHP code was reduced to the very minimum. We run and test code onClient platform: *chrome, firefox, IE.Html5,Css3,Jquery,Ajax,Bootstrap* are the Client techniques we used.Bootstrap is a free collection of tools for creating websites and web applications. It contains HTML and CSS-based design templates for typography, forms, buttons, navigation and other interface components, as well as optional JavaScript extensions.

2.ER diagram



3.Relational schema And Database table

Database normalization is the process of organizing the fields and tables of a relational database to minimize redundancy. Normalization usually involves dividing large tables into smaller (and less redundant) tables and defining relationships between them. The objective is to isolate data so that additions, deletions, and modifications of a field can be made in just one table and then propagated through the rest of the database using the defined relationships.

1NF is a property of a relation in a relational database. A relation is in first normal form if the domain of each attribute contains only atomic values, and the value of each attribute contains only a single value from that domain.

2NF:Functional dependencies of non-prime attributes on candidate keys are full functional dependencies.

3NF: No non-prime attribute is transitively dependent on a candidate key.

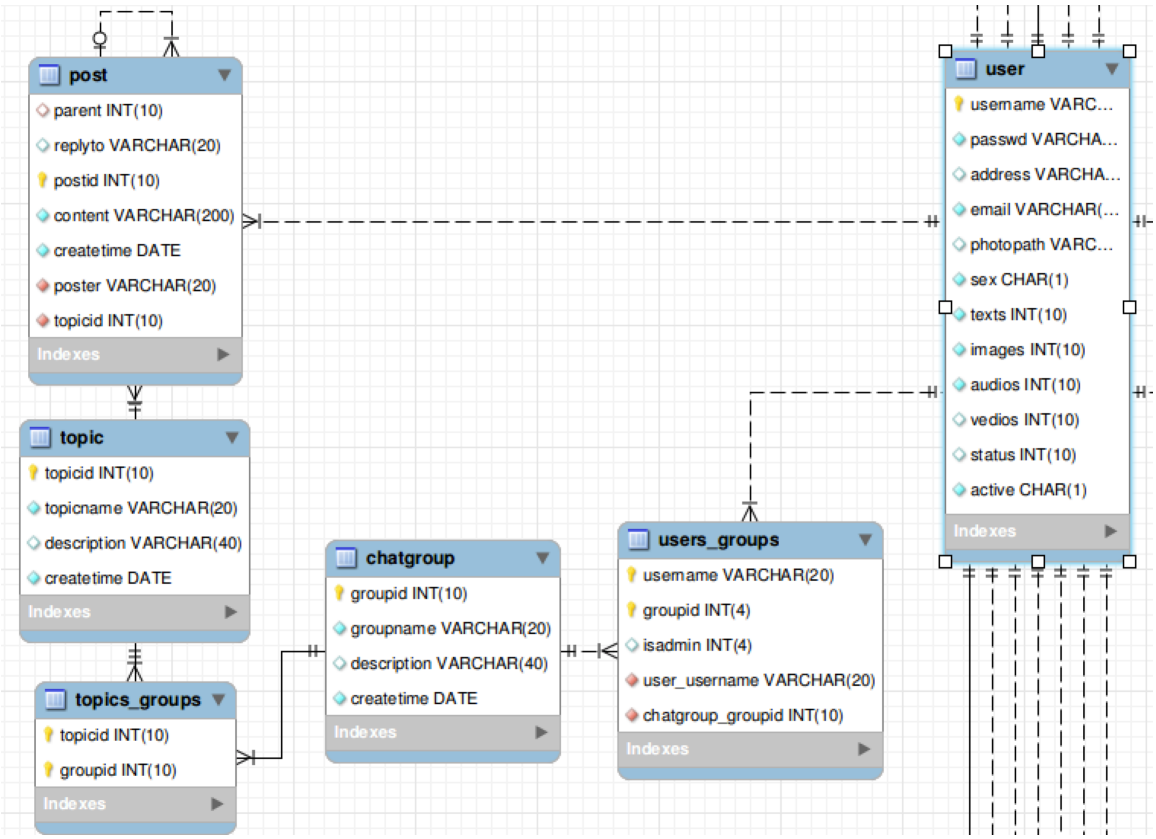
BCNF: A table is in Boyce-Codd normal form (BCNF) if and only if, for every one of its non-trivial functional dependencies X → Y, X is a superkey. i.e., X is either a candidate key or a superset thereof.

4NF： A table is in fourth normal form (4NF) if and only if, for every one of its non-trivial multivalued dependencies X -- >> Y, X is a superkey, i.e., X is either a candidate key or a superset thereof.

5NF： a 4NF table is said to be in the 5NF if and only if every join dependency in it is implied by the candidate keys.

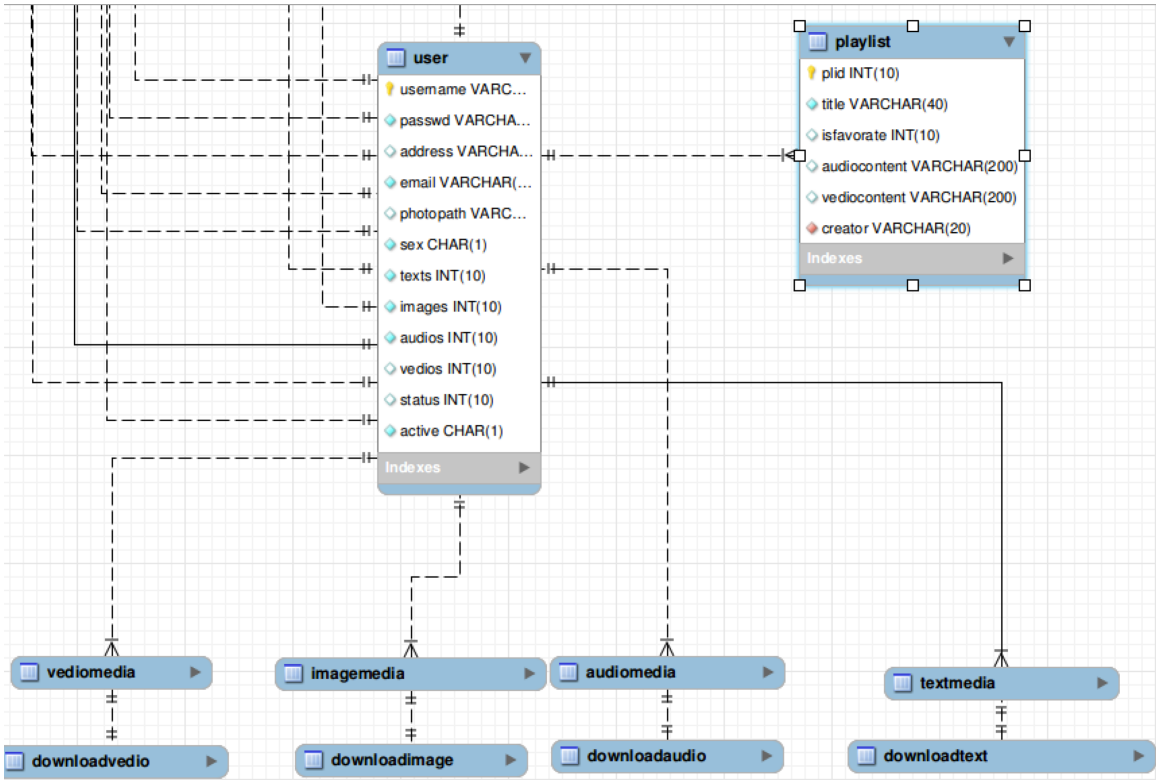
3.1 Discussion

`chatgroup` , `users_groups`, `topic` ,`topics_groups` ,` post user`



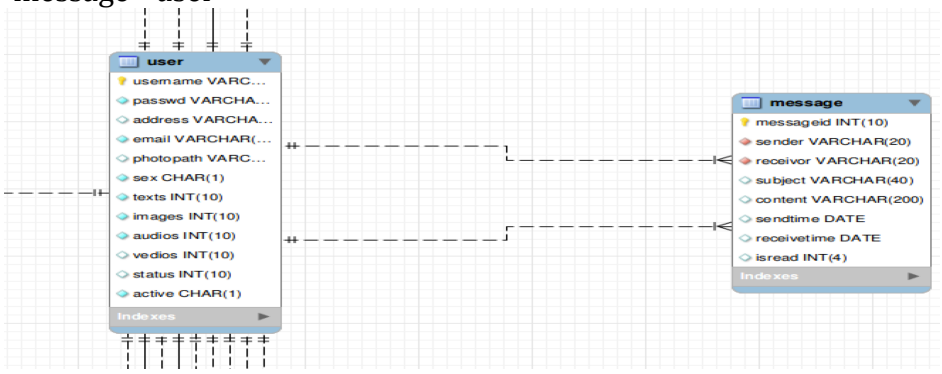
3.2playlist

`playlist` `user` `imagemedia` `audiomedia` `textmedia` `vediomedia`



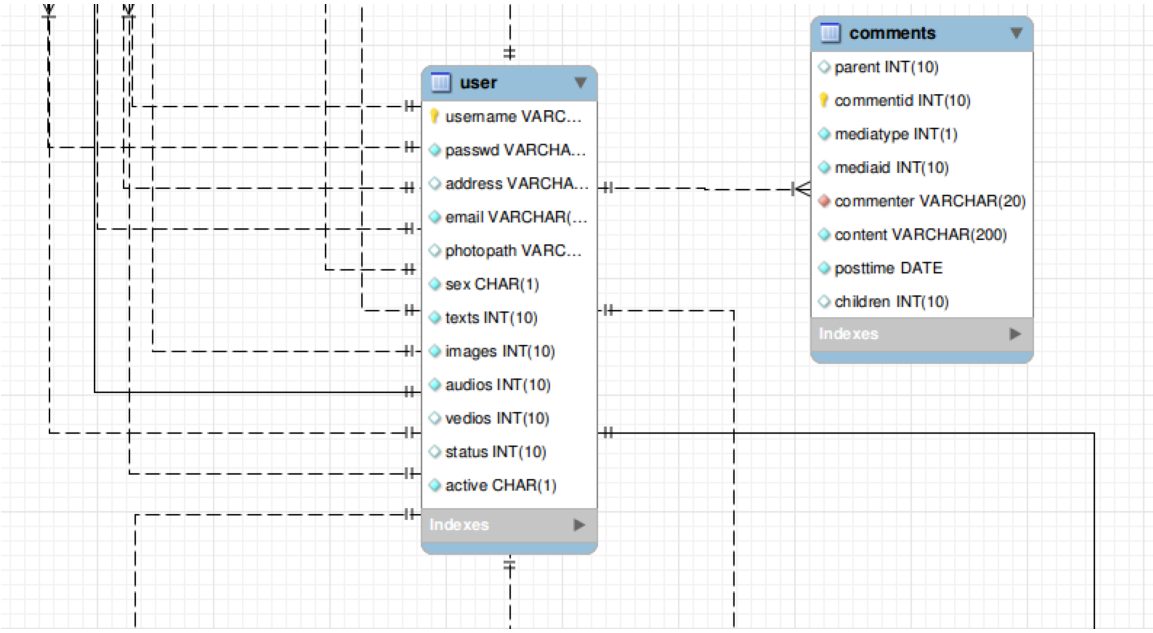
3.3 message

`message` `user`



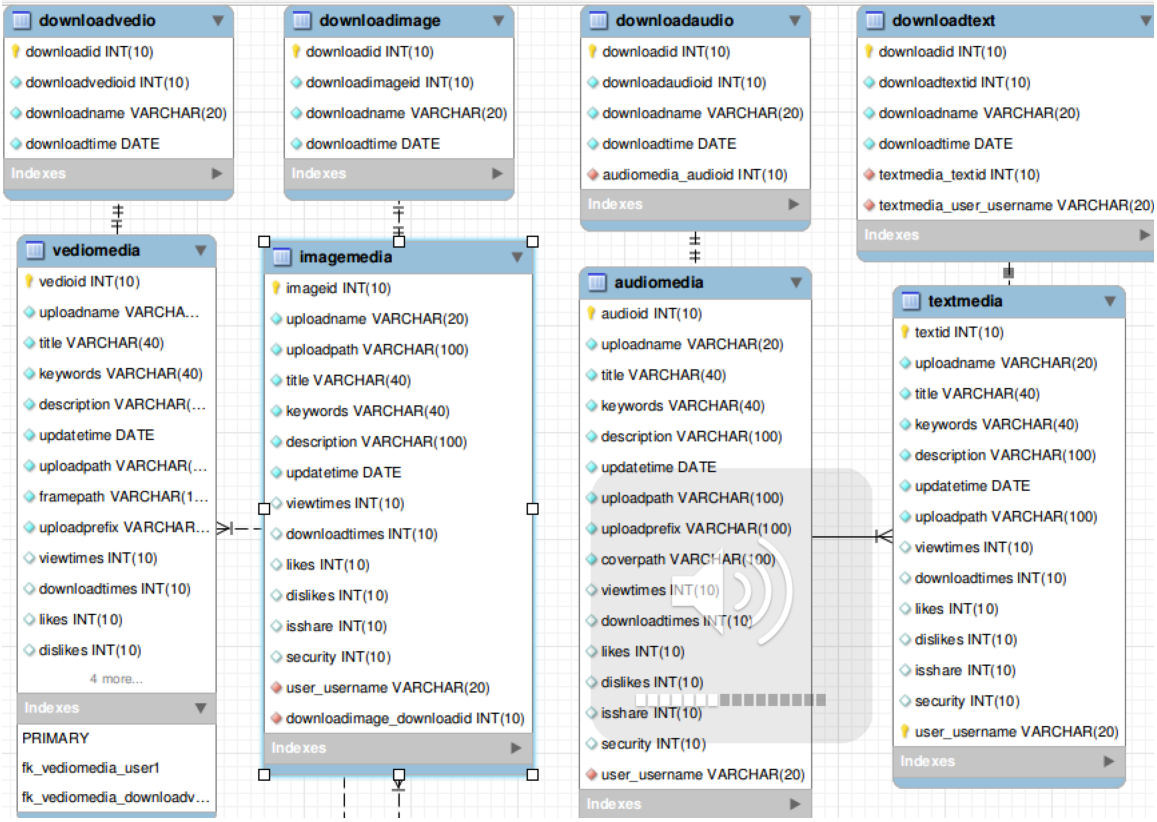
3.4 comments

`comments` `user` `imagemedia` `audiomedia` `textmedia` `vediomedia`



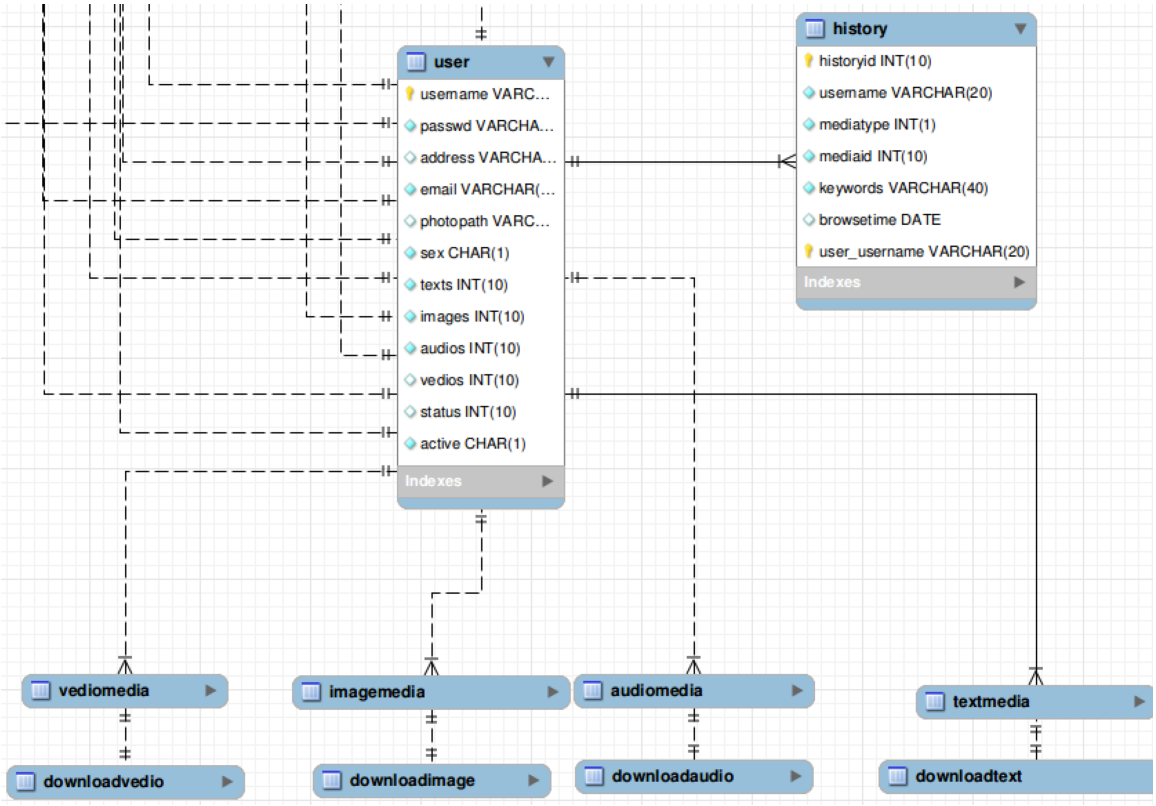
3.5 media

`user` `imagemedia` `audiomedia` `textmedia` `vediomedia`



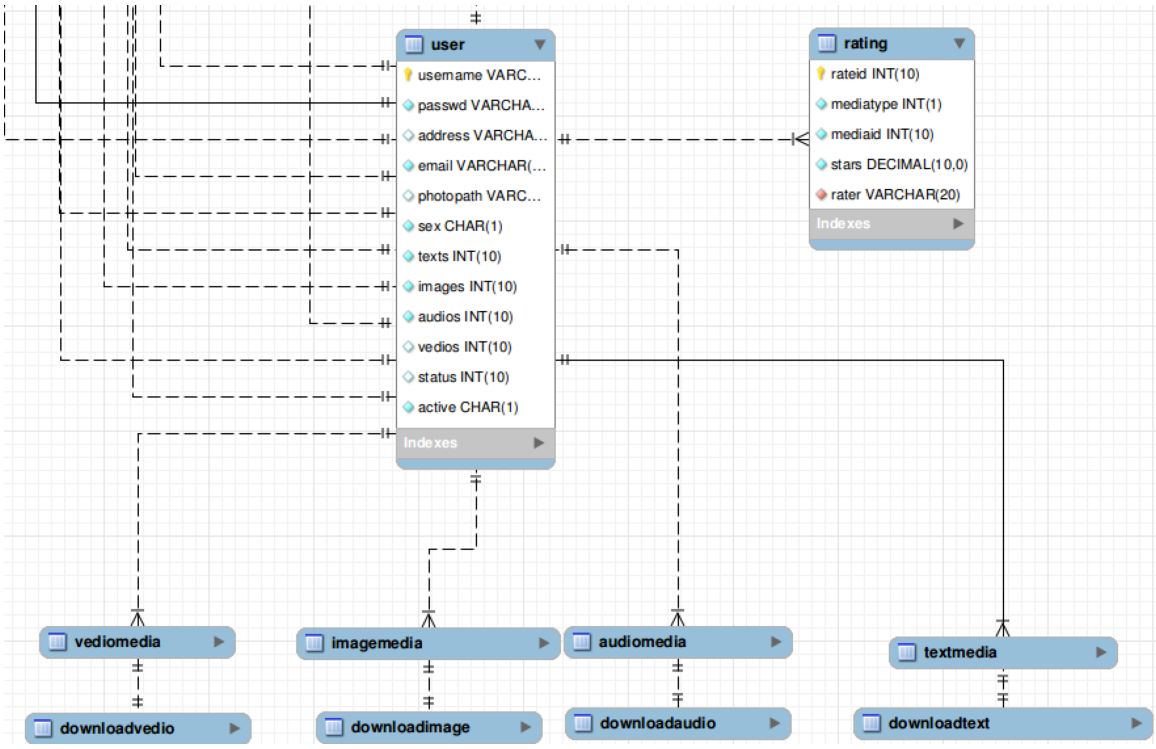
3.6 history

`history` `user` `imagemedia` `audiomedia` `textmedia` `vediomedia`



3.7 rating

`rating` `user` `imagemedia` `audiomedia` `textmedia` `vediomedia`



4 Database queries

4.1Doctrine Object-Relational mapper

Object relational mapper (ORM) for PHP that sits on top of a powerful database abstraction layer (DBAL). One of its key features is the option to write database queries in a proprietary object oriented SQL dialect called Doctrine Query Language (DQL), inspired by Hibernates HQL. This provides developers with a powerful alternative to SQL that maintains flexibility without requiring unnecessary code duplication.

Object ⇔ table

4.2 SQL Query based on ORM

Now list the four basic sql query based on ORM

4.2.1 Create:

user create:

```
$user = new User();
        $email = $request->get('email');
        $pos = strrpos($email, "@");
        $username = substr($email,0,$pos);
        $password = $request->get('password');
```

```
        $user->setUsername($username);
        $user->setEmail($email);
        $user->setPasswd($password);

        $em = $this->getDoctrine()->getManager();
        $em->persist($user);
        $em->flush();
```

4.2.2 Delete

Delete friend:

```
    $session = $request->getSession();
    $fromname = $session->get('uname');
    $em = $this->getDoctrine()->getManager();
    $to = $em->getRepository('MSSCoreBundle:Contactlist')-
>findOneBy(array('fromname'=>$fromname, 'toname'=>$toname));
    if($to){
      $em->remove($to);
    }
    $from = $em->getRepository('MSSCoreBundle:Contactlist')-
>findOneBy(array('fromname'=>$toname, 'toname'=>$fromname));
    if($from){
      $em->remove($from);
    }
    $em->flush();
```

4.2.3 Modify:

Update password:

```
$em = $this->getDoctrine()->getManager();
        $user = $em->getRepository('MSSCoreBundle:User')->find($username);
        if($user->getPasswd() == $request->request->get('oldpassword')){
          if($request->request->get('oldpassword') == $request->request-
>get('password')){
            $perror = 'No change!';
            return $this->render("MSSCoreBundle:User:updateprofile.html.twig",
array('type' => 'password', 'perror' => $perror));
          }else{
            $user->setPasswd($request->request->get('password'));
            $em->flush();
          }
        }else{
          $perror = 'Error old password!';
          return $this->render("MSSCoreBundle:User:updateprofile.html.twig",
array('type' => 'password', 'perror' => $perror));
        }
```

4.2.4 Query:

most view videos

```
$em = $this->getDoctrine()->getManager();
      $vquery = $em->createQueryBuilder()
            ->select('v.vedioid, v.title, v.uploadpath, v.keywords, v.framepath')
            ->from('MSSCoreBundle:Vediomedia', 'v')
            ->orderBy('v.viewtimes', 'DESC')
            ->setMaxResults(13)
            ->getQuery();
  $vresults = $vquery->getResult();
```

5.Metube Implementation Details

5.1 Media Organization

5.1.1Browse by category:

5.1.1.1 Browse by image

Starting from web pages, user sendsa*Image-Browsing* request, system searches matched routing *url[mss_core_cate_images]*, based on routing configuration file, the request is dispatched to Controller*MSSCoreBundle:Category*. Through method *imagesAction()* , we fetch all types of mediafrom table `*imagemedia*`, and then, send a response including all media results to user' s browser in page *Category/images.html.twig.*

5.1.1.2 Browse by text

Starting from web pages, user sends a*Text- Browsing* request, system searches matched routing *url[mss_core_cate_texts]*, based on routing configuration file, the request is dispatched to Controller*MSSCoreBundle:Category*. Through method *textsAction()* , we fetch all types of media from table `*textmedia*`, and then, send a response including all media results to user' s browser in .

5.1.1.3 Browse by audio

Starting from web pages, user sends a*Audio- Browsing* request, system searches matched routing *url[mss_core_cate_audios]*, based on routing configuration file, the request is dispatched to Controller*MSSCoreBundle:Category*. Through method *audiosAction()* , we fetch all types of media from table `*audiomedia*`, and then, send a response including all media results to user' s browser in page *Category/audios.html.twig.*

5.1.1.4 Browse by video

Starting from web pages, user sends a*Viedo- Browsing* request, system searches matched routing *url[mss_core_cate_videos]*, based on routing configuration file, the request is dispatched to Controller*MSSCoreBundle:Category*. Through method*videos*Action() , we fetch all types of media from table `*videomedia*`, and then, send a response including all media results to user' s browser in page *Category/videos.html.twig.*

5.1.2Channel

5.1.2.1 User's own channel

Starting from web pages, user sends a Show user's channels request, system searches matched routing url[mss_core_user_channels], based on routing configuration file, the request is dispatched to Controller MSSCoreBundle:User. Through method channelsAction() , we fetch specific type of channels that user requests from specific media table, and then, send a response including all channels of one user to user' s browser in page User/channels.html.twig.

5.1.2.2 User's subscribed channel

Starting from web pages, user sends a Show user's subscribed schannels request, system searches matched routing url[mss_core_user_subchannels], based on routing configuration file, the request is dispatched to Controller MSSCoreBundle:User. Through method subchannelsAction() , we fetch channels that user requests from table 'sbuscribedchannel', and then, send a response including all subscribed channels of a user to user' s browser in page User/subchannels.html.twig.

5.1.2.3 Subscribe channels

Starting from web pages, user sends a Subscribe channel request, system searches matched routing url[mss_core_user_subscribechannel], based on routing configuration file, the request is dispatched to Controller MSSCoreBundle:User. Through method subscribechannelAction() , we insert an entry to table 'subscribedchannel', and then, send a response including all his/her subscirbed channel channels results to user' s browser in page User/subchannels.html.twig.

5.1.2.4 Unsubscribe channels

Starting from web pages, user sends a Unsubscribe channel request, system searches matched routing url[mss_core_user_unsubscirbechannel], based on routing configuration file, the request is dispatched to Controller MSSCoreBundle:User. Through method unsubscribechannelAction() , we delete an specific entry from table 'subscribedchannel', and then, send a response including all his/her subscirbed channel channels results to user' s browser in page User/subchannels.html.twig.

5.1.3Playlists

5.1.3.1 Playlist-Checking

Starting from web pages, user sends a *Playlist-Checking* request, system searches matched routing *url[mss_core_list_play]*, based on routing configuration file, the request is dispatched to Controller*MSSCoreBundle:Medialist*. Through method*playAction()* , we fetch all types of media from table `*playlist* `, and then, send a response including all media results to user' s browser in page *Playlist/playlist.html.twig.*

5.1.3.2 Playlist-Creating

Starting from web pages, user sends a *Playlist-Creating* request, system searches matched routing *url[mss_core_list_create]*, based on routing configuration file, the request is dispatched to Controller*MSSCoreBundle:Medialist*. Through method*creatAction()* , we fetch all types of media from table `*playlist* `, and then, send a response including all media results to user' s browser in page *Playlist/playlist.html.twig.*

5.1.3.3 Favoritelist-Checking

Starting from web pages, user sends a *Favoritelist-Checking* request, system searches matched routing *url[mss_core_list_favorite]*, based on routing configuration file, the request is dispatched to Controller*MSSCoreBundle:Medialist*. Through method*favoriteAction()* , we fetch all types of media from table `*playlist* `, and then, send a response including all media results to user' s browser in page *Playlist/playlist.html.twig.* In favoriteAction(),there are two parameters,type and mediaid.Type means media type:mage,audio,video,text.Mediaid is the primary key to identify the specific media.

5.1.3.4 add Media playlist

In this part, user can add spedific media to current account.Starting from web pages, user sends a*Media-Adding* request, system searches matched routing *url[mss_core_list_addToPlay]*, based on routing configuration file, the request is dispatched to Controller*MSSCoreBundle:Medialist*. Through method*addToPlayAction()* , we fetch all types of media from table `*playlist* `, and then, send a response including all media results to user' s browser in page *Playlist/playlist.html.twig.* In *addToPlayAction()*there are four parametersused,type and mediaid,keywords,plid.Type means media type: mage, audio, video, text. Mediaid is the primary key to identify the specific media.Keywords is the simple description about the specific media. Plid identifies the playlist id,it helps to differentiate one media from another.In this part, Ajax is used, through which webpage will not be refreshed after processing the data.

5.1.3.5 Check media details

In this part, user can click one item to see more details about item. Starting from web pages, user sends aplaylist-listingrequest, system searches matched routing *url[mss_core_list_details]*, based on routing configuration file, the request is dispatched to Controller*MSSCoreBundle:Medialist*. Through method*detailsAction() with* parameter `*Plid*` that identifies the specific media.
we fetch all medias from table `*playlist* ` related to the given `plid`, and then, send a response including all media results to user' s browser in page *Playlist/details.html.list.*

5.1.4Most recently uploaded

In this part, a webpage is displayed, on which media upload most recently.The homepage is displayed this way bydefault.Before homepage is shown up on the front of user, system match routing *url[mss_core_homepage]*, based on routing configuration file, and then homepage request is dispatched to Controller *MSSCoreBundle:Default*. Through method*indexAction() in this controller*, We fetch the top 15 recently uploaded media from table `*imagemedia*`,`*textmedia*`,`*videomedia*`,`*audiomedia*` by sorting the upload time of

medias in decrease order. After that, send a response including all media results to user' s browser in page *Home/index.html.twig.*

5.1.5Most viewed

In this part, the homepageisdisplayed on which medias viewed most. Based on user's request, we convert the recently uploaded medias in homepage to most viewed medias.Before homepage is shown up on the front of user, system match routing *url[mss_core_homepage]*, based on routing configuration file, and then homepage request is dispatched to Controller *MSSCoreBundle:Default*. Through method*indexAction() in this controller*, We fetch the top 15 recently uploaded media from table `*imagemedia*`,`*textmedia*`,`*videomedia*`,`*audiomedia*` by sorting the upload time of medias in decrease order. After that, send a response including all media results to user' s browser in page *Home/index.html.twig.*

5.2 User interaction

5.2.1Messaging

This part helps user communicates with their friends."The badge nearby the menu 'inbox', and it shows how many incoming messages.

5.2.1.1Check message list

Before homepage is shown up on the front of user, system match routing *url[mss_core_homepage]*, based on routing configuration file, and then homepage request is dispatched to Controller *MSSCoreBundle:Default*. Through method*indexAction() in this controller*, We fetch the top 15 recently uploaded media from table `*imagemedia*`,`*textmedia*`,`*videomedia*`,`*audiomedia*` by sorting the upload time of medias in decrease order. After that, send a response including all media results to user' s browser in page *Home/index.html.twig.*
, user sends a *Message-listing* request, system searches matched routing *url[mss_core_message_list]*, based on routing configuration file, the request is dispatched to Controller *MSSCoreBundle:Message*. Through method*getAllAction()* , we fetch datafrom table ` *message*`related to current login user, and then, send a response including all media results to user' s browser in pageMessage/messageList.html.twig

5.2.1.2 Create Message

Before homepage is shown up on the front of user, system match routing url[mss_core_homepage], based on routing configuration file, and then homepage request is dispatched to Controller MSSCoreBundle:Default. Through method indexAction() in this controller, We fetch the top 15 recently uploaded media from table ` imagemedia`,`textmedia`,`videomedia`,`audiomedia` by sorting the upload time of medias in decrease order. After that, send a response including all media results to user' s browser in page Home/index.html.twig. *User sendsa Message-Creating request, system searches matched routing url[mss_core_message_creat], based on routing configuration file, the request is dispatched to Controller MSSCoreBundle:Message. Through methodcreatAction()*We create a

new message and insert it into `message`, send a reponse to notice user's browser that message is successifully created.

5.2.1.3 Read Message

Before homepage is shown up on the front of user, system match routing *url[mss_core_homepage]*, based on routing configuration file, and then homepage request is dispatched to Controller *MSSCoreBundle:Default*. Through method *indexAction() in this controller*, We fetch the top 15 recently uploaded media from table ` *imagemedia*`,`*textmedia*`,`*videomedia*`,`*audiomedia*` by sorting the upload time of medias in decrease order. After that, send a response including all media results to user' s browser in page *Home/index.html.twig.*when user start reading a new coming message, it will send a asynchronous request*„ system searches matched routing url[mss_core_message_read], based on routing configuration file, the request is dispatched to Controller MSSCoreBundle:Message. Through methodreadAction()*with parameter of messageid, We update the message related to this messageid to set its isread status as 1, which means we have already read this message, and then, send a reponse to notice user's browser that message is successifully updated.

5.2.1.4 Reply message

Before homepage is shown up on the front of user, system match routing url[mss_core_homepage], based on routing configuration file, and then homepage request is dispatched to Controller MSSCoreBundle:Default. Through method indexAction() in this controller, We fetch the top 15 recently uploaded media from table ` imagemedia`,`textmedia`,`videomedia`,`audiomedia` by sorting the upload time of medias in decrease order. After that, send a response including all media results to user' s browser in page Home/index.html.twig*. user sends a Message-Replying request, system searches matched routing url[mss_core_message_reply], based on routing configuration file, the request is dispatched to Controller MSSCoreBundle:Message. Through methodreplyAction() with parameter of messageid,* We create a new replied message and insert it into `message`, at the same time , set its parent to messageid , and then send a reponse to notice user's browser that message is successifully replied.

5.2.2 Commenting

requirement: user

List all comments related to a particular media

5.2.2.1 Create Comment:

Before homepage is shown up on the front of user, system match routing url[mss_core_homepage], based on routing configuration file, and then homepage request is dispatched to Controller MSSCoreBundle:Default. Through method indexAction() in this controller, We fetch the top 15 recently uploaded media from table ` imagemedia `,`textmedia `,` videomedia `,` audiomedia ` by sorting the upload time of medias in decrease order. After that, send a response including all media results to user' s browser in page Home/index.html.twig.We create a new comment and insert it into `comments`, send a reponse to notice user's browser that message is successifully created.

5.2.2.2 Reply Comment

Through methodreplyAction() with parameter of messageid, We create a new replied message and insert it into `message`, at the same time , set its parent to messageid , and then send a reponse to notice user's browser that message is successifully replied.

5.2.3 Media rating

When user view a specific media, and change the number of starts related to this media, it will send a asynchronous request to server, *system searches matched routing url[mss_core_media_rate], based on routing configuration file, the request is dispatched to Controller MSSCoreBundle:Rating. Through methodnewRateAction(), we update the rated starts related to that media to set the new numbers of start in `rating` table.,and then,*send a reponse to notice user's browser that reating is successifully updated.

5.2.4 Group discussion

5.2.4.1 Check discussion list

In homepage, user sends aGrouplist-Listing request, system searches matched routing url[mss_core_discussion_listg], based on routing configuration file, the request is dispatched to Controller MSSCoreBundle:Discussion. Through methodlistGroupAction() , we fetch datafrom table `chatgroup `, and then, send a response including all media results to user' s browser in page listGroup.html.twig.

5.2.4.2 Create discussion

We create a new group and insert it into `chatgroup` ,`users_groups`
, send a reponse to notice user's browser that new group is successifully created.

5.2.4.3 Topic-list Check

In homepage, user sends a *Topic-Lisitngrequest, system searches matched routing url[mss_core_discussion_listt], based on routing configuration file, the request is dispatched to Controller MSSCoreBundle:Discussion. Through methodlistTopicAction() , we fetch datafrom table `topic`, `topics_groups`, and then, send a response including all media results to user' s browser in page listTopic.html.twig.*

5.2.4.4 Create discussion topic

We create a new discussion topic and insert it into `topic`, send a reponse to notice user's browser through *page createTopic.html.twig* that topic is successifully created.

5.2.4.5 Post List Checking

Under a topic in a group, user sends a post-listing request. System searches matched routing url*[mss_core_discussion_listp], based on routing configuration file, the request is dispatched to Controller MSSCoreBundle:Discussion. Through methodlistPostAction() , we fetch datafrom table `post `, and then, send a response including* all post results for given topicid and groupid*to user' s browser in page listPost.html.twig.*Put this sentence after "method listPostAction()"

5.2.4.6 Create New Post

Before homepage is shown up on the front of user, system match routing url[mss_core_homepage], based on routing configuration file, and then homepage request is dispatched to Controller MSSCoreBundle:Default. Through method indexAction() in this controller, We fetch the top 15 recently uploaded media from table ` imagemedia `,`textmedia `,` videomedia `,` audiomedia ` by sorting the upload time of medias in decrease order. After that, send a response including all media results to user' s browser in page Home/index.html.twig.We create a new post and insert it into `post`, send a reponse to notice user's browser that message is successfully created.

5.2.4.7 Reply Post

We create a new reply and insert it into ` post `, send a reponse to notice user's browser through page'*listPost.html.twig'that message is successfully created.*
In homepage, user sends aPost-Replying request, system searches matched routing url[mss_core_discussion_replyp], based on routing configuration file, the request is dispatched to Controller MSSCoreBundle:Discussion. Through methodreplyPostAction() , we fetch datafrom table `post `, and then, send a response including all media results to user' s browser in page listPost.html.twig

5.3 Search

5.3.1 Keywords-based search

*In homepage, user sends aKeywords-based Searching request*including the keywords*, system searches matched routing url[mss_core_discussion_ksearch], based on routing configuration file, the request is dispatched to Controller MSSCoreBundle:Search.Through methodkeySearchAction().* keySearch()has one parameter 'key'*. It identifies the keywords of all media. We fetch datafrom table `imagemedia `,`audiomedia`,`videomedia`,`textmedia`,*and then, send aresponse including all media related to those keywordsto user' s browser in page *Search/searchList.html.twig.*keySearch()has one parameter 'key'*. It identifies the keywords of all media.*

5.3.2 Word cloud

In homepage, user sends acloudkey-Searching requestincluding a cloud keywords , system searches matched routing url[mss_core_discussion_cksearch], based on routing configuration file, the request is dispatched to Controller MSSCoreBundle:Search. Through methodcloudkeySearchAction().cloudkeySearchAction ()has one parameter 'key'*. It identifies the keywords of all media. We fetch datafrom table `imagemedia `,`audiomedia`,`videomedia`,`textmedia`,*and then, send aresponse including all media related to cloud keywords to user' s browser pagesin*Home/index.html.twig.* How do we get cloud

keywords? We search most popular keywords searched by users in `history` table, return top searched keywords as key in our cloud..

5.3.3 Media recommendation

*After user login to our website, the recommended media will be listed on the homepage, so before user enter to homepage, system matched routing url, user sends a Media-Recommendation request, system searches matched routing url[mss_core_homepage], based on routing configuration file, the request is dispatched to Controller MSSCoreBundle:Default. Through methodindexAction() ,We search most popular keywords searched by current user based on `history` table, by take advantage of this keywords, we return the top 15 popular medias related to those keywords to user' s browser in we fetch datafrom table `history `,*and then, send aresponse including all media results to user' s browser in *Home/index.html.twig.*

5.3.4 Feature-based media search

In homepage, user sends a *Media-Searching request,*feature-based media searching request including feature based keywords,system searches matched routing*url[mss_core_media_fsearch],* based on routing configuration file, the request is dispatched to Controller*MSSCoreBundle:Default. Through methodfeatureSearchAction().*We fetch datafrom table `imagemedia `,`audiomedia`,`videomedia`,`textmedia`,and then, send aresponse including all media related to feature-based keywords to user' s browser pages Home/fsearchList.html.twig.

5.3.5User Account

5.3.5.1 Registration

In sign-in page, user sends a sign-up request, system searches matched routing url[mss_core_user_signup], based on routing configuration file, the request is dispatched to Controller MSSCoreBundle:User. Through method signupAction() , it sends a response to user' s browser in page User/signup.html.twig. Then user can input some information and click the sign up button to finish registration also through method signupAction(). This time, it will insert an new entry in table 'user' and send an email with an active link to the given email address then sends a response to user's browser in page User/inactive.html to indicate that a new user needs to active her/her account.

5.3.5.2 Activate

User need to go to her/her email account inbox to find the active email and click the lick in the email content, this will send go to the active web page User/active.html.twig. Then user can upload a profile photo then click the upload button to complete the active action, system searches matched routing url[mss_core_user_active], based on routing configuration file, the request is dispatched to Controller MSSCoreBundle:User. Through method uploadAction(), it will update the 'active' value in table 'user' and insert an entry to table 'imagemedia', then sends a response to user's browser in page Home/index.html.twig.

5.3.5.3 Sign in

In sign-in page, user sends a sign-in request, system searches matched routing url[mss_core_user_signin], based on routing configuration file, the request is dispatched to Controller MSSCoreBundle:User. Through method signinAction() , it will update the 'state' value in table 'user' and set some 'key=>value' in a new session, then it sends a response to user's browser in page Home/index.html.twig.

5.3.5.4 Profile Update

In homepage, user sends a show profile request, system searches matched routing url[mss_core_user_profile], based on routing configuration file, the request is dispatched to Controller MSSCoreBundle:User. Through method profileAction(), it sends a response to user's browser in page User/account.html.twig. In this page, user can choose to change password, profile photo or all other information. To any change request that user sends, system searches matched routing url[mss_core_user_updateprofile], based on routing configuration file, the request is dispatched to Controller MSSCoreBundle:User. Through method updateprofileAction(), it will update corresponding value or values in table 'user', then it sends a response to user's browser in page User/account.html.twig in order to show the newest profile information to user.

5.3.5.4 Contact

In page User/account.html.twig, user sends a show friends request, system searches matched routing url[mss_core_user_friends], based on routing configuration file, the request is dispatched to Controller MSSCoreBUndle:User. Through method friendsAction(), it sends a respond to user's browser in page User/friends.html.twig.
In page User/account.html.twig, user sends a show foes request, system searches matched routing url[mss_core_user_foes], based on routing configuration file, the request is dispatched to Controller MSSCoreBUndle:User. Through method foesAction(), it sends a respond to user's browser in page User/foes.html.twig.

5.3.5.5 Friend/foe

In page User/friends, user can view all the information related friends management. For example, all the friends, all the friends request that other users send, all the friend request that the current user sends and all the blocked users. User can send a lot of requests in this page. Such as cancel, refuse or confirm a friend request, make friend or unfirend with a user, make foe with a user, block or unblock a user. All this request, system searches matched routing url[mss_core_user_canlefriend], url[mss_core_user_refusefriend], url[mss_core_user_confirmfriend], url[mss_core_user_makefrind], url[mss_core_user_unfriend], url[mss_core_user_makefoe], url[mss_core_user_block], url[mss_core_user_unblock], based on routing configuration file, the request is dispatched to corresponding controller to do corresponding action, all these controllers will sends a respond to user's browser in page User/friends.html.twig to show the newest information about friend.
In page User/foes, user can view all the information related foes management. User can send two requests, unfoe or make friend with a foe. All this request, system searches matched routing url[mss_core_user_unfoe], url[mss_core_user_makefriend], based on routing configuration file, the request is dispatched to corresponding controller to do corresponding action, all these controllers will sends a respond to user's browser in page User/foes.html.twig to show the newest information about foes.

Of course, in other web pages, user can make friend, unfriend, make foe, unfoe, block, unblock with other users through the same routing and controller.

5.3.5.6 User blocking

Starting from web pages, user sends a block user request, system searches matched routing url[mss_core_user_block], based on routing configuration file, the request is dispatched to Controller MSSCoreBundle:User. Through method blockAction() , it sends a response to user's browser in page User/block.html.twig. In this page, user can send a block user with specific type request, system searches matched routing url[mss_core_user_blocktype], based on routing configuration file, the request is dispatched to Controller MSSCoreBundle:User. Through method blcoktypeAction(), it will insert an entry to table 'contactlist' and then sends a response to user's browser in page User/friends.html. Starting from web pages, user sends a unblock user request, system searches matched routing url[mss_core_user_unblock], based on routing configuration file, the request is dispatched to Controller MSSCoreBundle:User. Through method unblockAction() , it will delete an entry from table 'contactlist' or just update certain value in table 'contactlist' and then sends a response to user's browser in page User/friends.html.

5.3.6 Data Sharing

A sign-in user can upload four different types of media to the system and download medias in the system. Each media that he/she uploads, he/she can set the security degree to private or public. For private medias, only friends can view and download them. For public medias, everyone can view and download them.

5.3.6.1 Upload

Starting from web pages, user sends a upload image request, system searches matched routing url[mss_core_media_uploadImage], based on routing configuration file, the request is dispatched to Controller MSSCoreBundle:Media. Through method uploadImageAction() , it will insert an entry to table 'imagemedia' and send a response to user's browser in page User/channels.html.twig.

Starting from web pages, user sends a upload text request, system searches matched routing url[mss_core_media_uploadText], based on routing configuration file, the request is dispatched to Controller MSSCoreBundle:Media. Through method uploadTextAction() , it will insert an entry to table 'textmedia' and send a response to user's browser in page User/channels.html.twig.

Starting from web pages, user sends a upload audio request, system searches matched routing url[mss_core_media_uploadAudio], based on routing configuration file, the request is dispatched to Controller MSSCoreBundle:Media. Through method uploadAudioAction() , it will insert an entry to table 'audiomedia' and send a response to user's browser in page User/channels.html.twig.

Starting from web pages, user sends a upload video request, system searches matched routing url[mss_core_media_uploadVideo], based on routing configuration file, the request is dispatched to Controller MSSCoreBundle:Media. Through method uploadVideoAction() , it will insert an entry to table 'videomedia' and send a response to user's browser in page User/channels.html.twig.

5.3.6.2 Meta data input

In upload page, user must input some meta data together with select a media to upload.

5.3.6.3 Download/View

Starting from web pages, user sends a download media request, system searches matched routing url[mss_core_media_download], based on routing configuration file, the request is dispatched to Controller MSSCoreBundle:Media. Through method downloadAction() , it will insert an entry to table according to media type, 'downloadtext', 'downloadimage', 'downloadaudio', 'downloadvideo' and download the media to user's computer.

5.3.6.4 Media sharing/blocking

When a user upload a media, she/he has the chance to select the security level for the media, either public or private. For media blocking, a user can block any user to view or download his/her medias.

6. Test

6.1 Unit Test

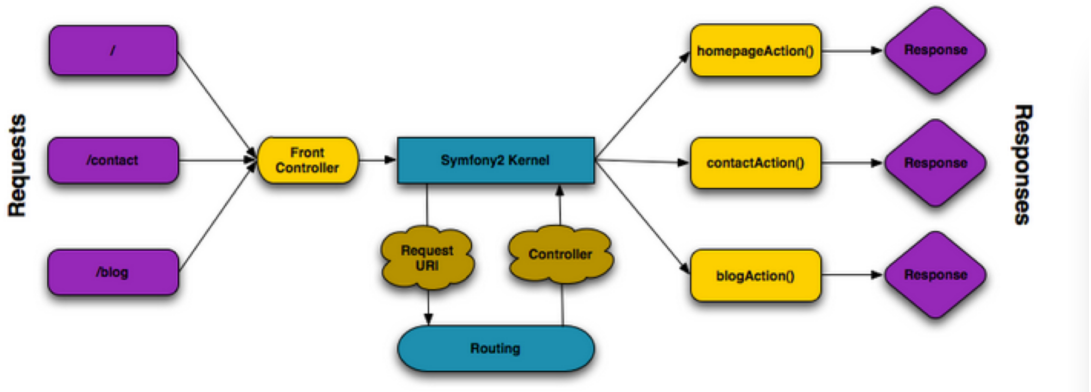
(1) For sever side, based on the different methods in different controllers, we manually test the corresponding sqls if they are successivefully perform and return the expected results
(2) For client side, we artificially create the presenting data and render them on the web pages, and we check if the data are correctly displayed on the webpages.

6.2 Integretion Test

(1) Between our controllers and mysql database, we test our controller if fetch data from related tables we desired.
(2) Among our controllers, we test if each controller is separate from each other during processing. For intersected function, we test the final data we get is correct.
(3) Between our controllers and view pages, we test requests from web pages if are correctly dispatched to requested controller method based on the routing configuration file.

7. System Design

Symfony framework is used to formulate this project's framework.



Incoming requests are interpreted by the routing and passed to controller functions that return Response objects.
Each "page" of your site is defined in a routing configuration file that maps different URLs to

different PHP functions. The job of each PHP function, called a controller, is to use information from the request - along with many other tools Symfony makes available - to create and return a Response object. In other words, the controller is where your code goes: it's where you interpret the request and create a response.

It's that easy! To review:

• Each request executes a front controller file;

• The routing system determines which PHP function should be executed based on information from the request and routing configuration you've created;

• The correct PHP function is executed, where your code creates and returns the appropriate Response object.