System Design Document

Chocolate Thunder

Jacob Steele
Sam Pourcyrous
Zubair Baig
Syeda Arafa Sulaiman

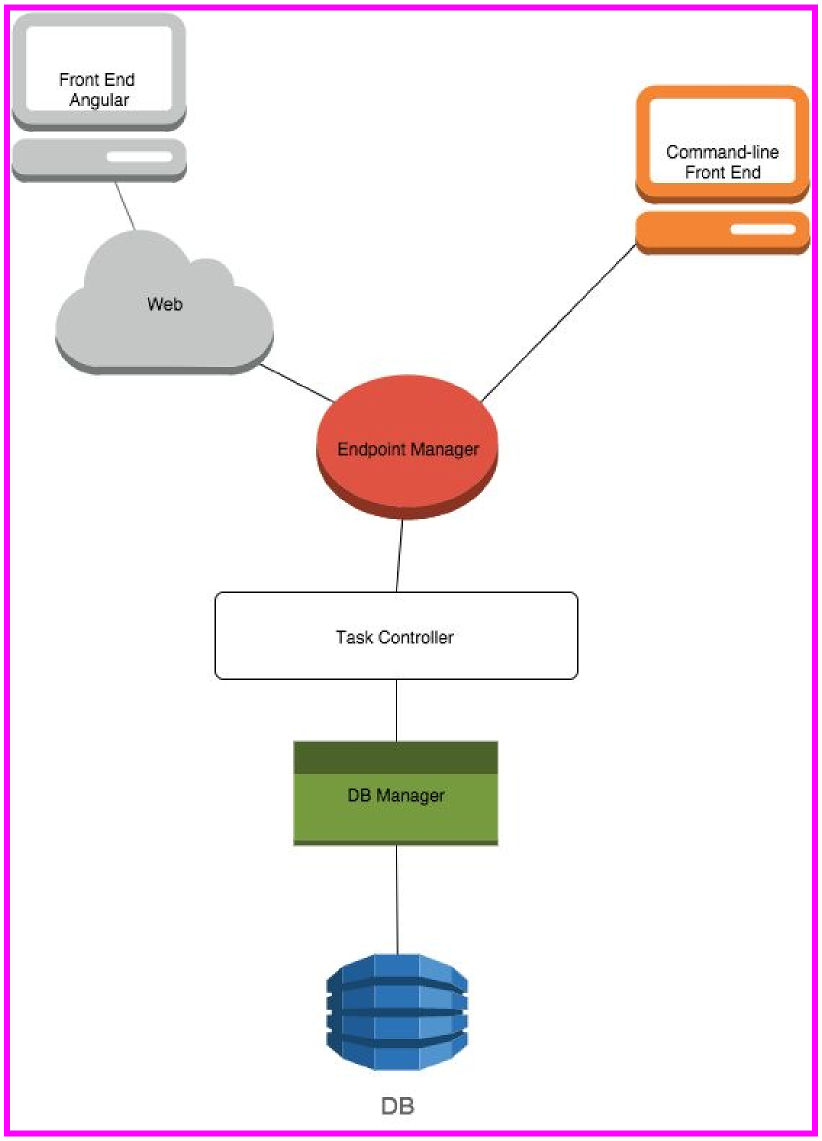
Table of Contents

System Interaction

For system interaction, our team has decided to use Apache Tomcat as the main platform for our web server. Our choice of platform stems from the fact that most of the website will consist of Java code which will encompass Java Servlets and JavaServer Pages. The platform will be coupled with Ubuntu 13.04.3 LTS 64-bit as our main operating system. We've chosen to go with Ubuntu, because it is quite user friendly for us as developers and does not require licensing. In addition to this, it is easy to setup Ubuntu virtual machines for us to use as base configurations for our web server, thereby ensuring every team member is on the same page. Since our main programming language will be Java, the main IDEs we'll be using are Eclipse and Intellij. These two IDEs are extremely versatile since they both have add-ons that allow integration with Maven and BitBucket. With Git being integrated into our IDEs, this is how we'll be distributing our code between each other. We will also use Maven to compile our code with any other external libraries we may be using for Java. The external libraries we're looking into using are JUnit for testing, Apache Hadoop for distributed processing, Mockito for automated unit tests, Apache Lucene for use in our search engine, Dropbox Core API for uploading files, Apache OpenNLP for language processing, and Apache Commons Email for verifying emails and sending verification emails to users. Finally, for our backend we're planning on using PostgreSQL in conjunction with JDBC to write and read information to the database.

For our front end, we've decided to use AngularJs which we will be using for our front end framework, as well Angular Bootstrap for easy front end templates. Since we're using AngularJs and Angular Bootstrap, this means we will also using HTML, JavaScript, and CSS.

Software Architecture



The software architecture of the website will comprise of a front-end using the AngularJS framework (accessible through a browser) and a command-line front-end (solely meant to offer access to administrators). Any HTTP GET or POST requests from the front-end will be sent to a java servlet that functions as an endpoint manager, consisting of a multitude of endpoints. The endpoint manager will determine which endpoint has been hit, and which task is to be carried out. Requests are subsequently routed to the relevant controller in the task controller layer, which may interact with the database manager. The database manager will query or manipulate the website's database as needed.

Dealing with Errors and Edge Cases

We have devised a plan to ensure errors are properly dealt with on our platform to enhance the end user's experience. Our first priority is to make sure we sanitize inputs that are submitted to our server. This will help guarantee that the correct, intended data will be returned to the user. Parsing submitted information will reduce errors returned in the response data and lower the chances of having our server compromised. Although server security is paramount, it is not the chief concern in the vision we have for our web application. We do, however, want to address the common security vulnerabilities websites have to help minimize concerns our users might have when using the website.

With every website, there is always a possibility that a user might access a restricted page or a page that doesn't exist. To tackle this situation, we will create landing pages for errors. We will have a custom 400-level error page and 500-level error page for client and server errors, respectively. These pages will notify the users that something went wrong and will prevent any confusion in case a scenario like this occurs.

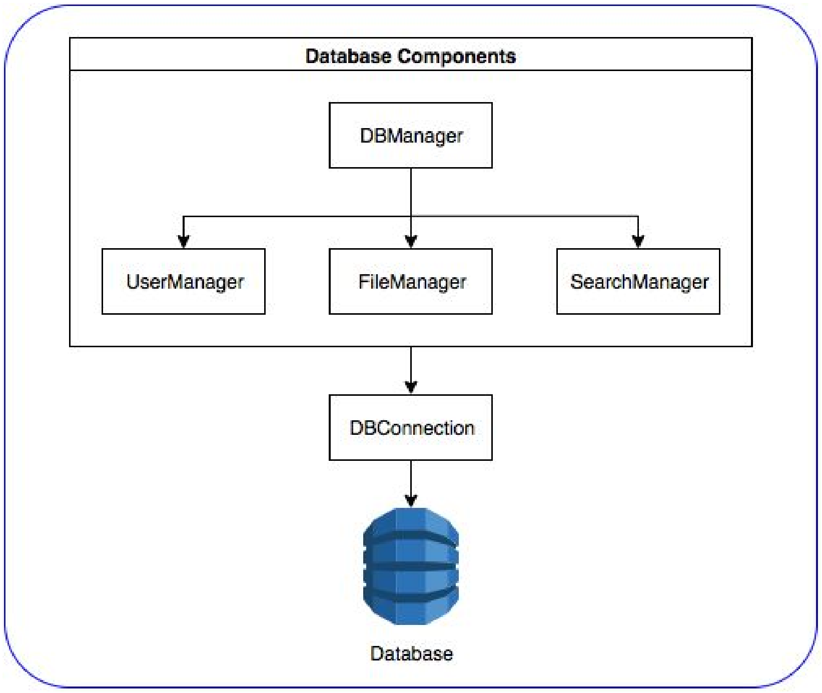
We have discussed some corner cases that we feel need to be considered. Our platform will show content based on the type of user. In our case, the user can either be a student or instructor. The kind of functionality available to visiting users is subject to the type of account the user possesses. We will be checking the session information to consistently validate the type of user throughout the website. This will prevent a student from performing instructor-level tasks. Since all users have the capability of uploading content to our server, we also need to consider the types of files a user can upload. Our software will check the extension of the file that is being uploaded and will only allow file types that are authorized for that type of user. This will help minimize useless files that will take up space on our server and will avoid malicious scripts from being uploaded. To further prevent unreliable information being stored on the website, we will grant instructors the permission to delete files they feel are not suitable for their respective course.

System Decomposition

Database Layer

The overall system architecture can provide perspective when diving into a detailed analysis of the architecture. The database layer itself represents the physical tables of data representing users and documents added to the site. However, no single Java class will interact directly with this layer except the DBManager class. This class is designed to act as the first layer in the separation of management when looking at overall site functionality. DBManager will not deal with any business logic but merely provide wrapper-like functionality to its children: FileManager, UserManager and SearchManager. The DBConnection class will merely serve as the database connection agent between the database layer and the database on the cloud.

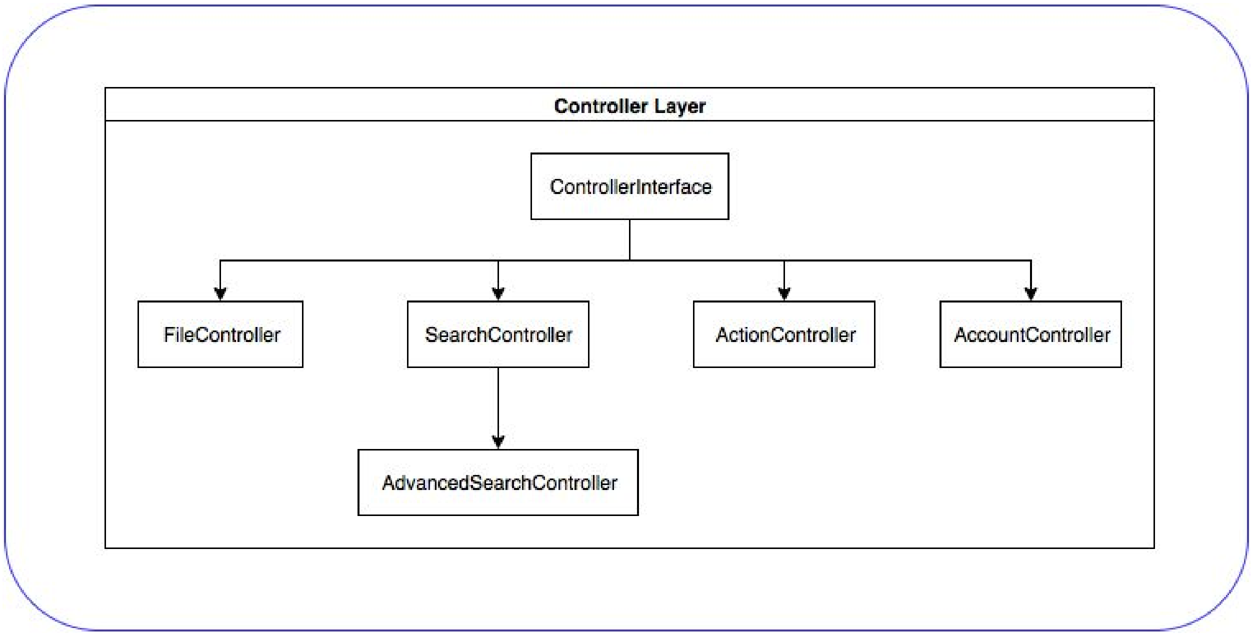
FileManager, UserManager and SearchManager represent the separation of different aspects of the search on the website. UserManager is responsible for all database queries related to users like creating and storing new accounts, responding to logins and providing data to construct User objects. FileManager similarly manages all tasks related to File objects like storing a newly uploaded file, tagging a file, approving a file and returning location of stored files. Search Manager is most likely going to have the largest chunk of business logic for database management since it will contain code for joining tables and searching across multiple dimensions like tags, course names, professor names etc.



Controller Layer

Moving up in the architectural diagram, the database layer interacts with the controller layer, which contains all of the business logic for the web program. To begin with, all of the individual controllers inherit from the ControllerInterface which contains a single (for now) stubbed method to parse json data pertinent to each controller. Even though at this time, that seems to be the only common functionality among controllers, we envision this to increase as the project grows in complexity. Thus, each controller will implement this parsing in their own way. The specific functionality for each controller is listed in the CRC cards but generally, each controller will be responsible for reading the payload sent in from POST requests after the EndPointManager calls the pertinent one. At this point, the SearchController is assumed to have the most business logic since it will contain all the Apache Lucene code dealing with search functionality.

As mentioned in the paragraph above, the SearchController along with the SearchManager in the DB layer will form the back end search engine. Lastly, we envision the AdvancedSearchController to be a child of the SearchController because AdvancedSearchController is a type of SearchController. It will contain specific search field delimiters that may be turned on or validated by the front end and then used in congruence with the SearchManager to search based on specific conditions.



Endpoint Manager

The last component of the web server’s back end logic will contain the EndpointManager class which, as the name implies, contains all of the end points for the web application. Each endpoint will be served on a specific url and after some preliminary sanity checks for proper request data structure, the code served by that endpoint will call the pertinent controller mentioned above.

Command Line Interface

The command line interface will offer administrators the ability to bypass the front end and invoke calls on the individual controllers directly. It will contain specific allowable API calls that will accept certain parameters and files (in the case of uploads) and perform its own sanity checks that the EndpointManager would otherwise be doing from the front end. At this point in the project, the group has committed to support batch file upload and search functionality from the command line, in addition to being able to delete users and documents. This list may grow as the project progresses. The result from the API calls will be forwarded to the commandline as a simple text result for the user to see.

CRC Cards

The CRC cards are found on the following page.

DBManager	FileManager, UserManager, DBConn
• Validates queries to database • Filters queries to database • Calling FileManager or UserManager	

FileManager	DBManager
• Add file to database • Remove file from database • Modify a file that is already in the database • Reads file from database • Add tag to file • Removes tag from file • Reads files allocated to tag	• DBConnection

UserManager	DBManager
• Adds users to database • Removes users from database • Modifies info for users in database • Reads user from database	• DBConnection

DBConnection	
• Knows how to connect to a database • Knows database connection URL	

SearchManager	DBManager
• has all of the dimensions allowed in an advanced search • knows how to perform an SQL query based on multiple dimensions (across tables) • knows how to perform an SQL query based on specific dimensions	• DBConnection

AccountController	
• Knows how to add account • Knows how to delete account • Knows how to verify account • Logs in the user	• DBManager • UserClass

AccountClass	
• Knows name • Knows account creation date • Knows files uploaded by user • Knows if user is student or instructor	

ActionController	
• Knows how to follow instructors • Knows how to add courses • Knows how to approve files	• DBManager

FileClass	
• Knows filename • Knows filesize • Knows who uploaded the file • Contains tags related to file • Knows which instructed approved it • Knows when the approval date was • Knows the file extension	

FileController	
• Parse file information	• DBManager
• Index content of file	• FileClass
• Knows how to add files	
• Knows how to delete files	
• Knows how to add tags to files	
• Knows how to upload batch files	
• Knows how to index batch files	

SearchController	AdvancedSearchController
• knows the current search term	• DBManager
• knows how to search across different dimensions	
• provides top results based on current search term	
• provides snippets of the contents of most relevant results	
• knows how to connect with google api	
• knows how to do a general internet search	

AdvancedSearchController	SeachController
• Has advanced search criteria	• DBManager
• Knows how to filter search based on criteria	

EndPointController	
• reroutes user responses to correct controller classes	• ActionController, FileController, SearchController, AccountController

AdminController	
• Admin can batch upload files • Admin can batch upload files to be indexed • Admin can view website analytics • Admin can delete files • Admin can delete accounts	• DBManager