University Student Market Project

December 6, 2015

XXX XXX

Table of Contents

Introduction

The University Student Market Project, henceforth referred to as CryptoCampus or its URL http://cryptocampus.com, provides an easy-to-use and private space for members of an academic community to exchange goods and services. For this project, a proof-of-concept demo website was developed specifically supporting users with an @lsu.edu email address. Only users with an @lsu.edu email address are allowed to register and when registered users sign in they are shown the private content at http://lsu.cryptocampus.com. From there users can view and post listings on category pages for goods and services. Users can also visit the profiles of other users to add reviews and see their reputation and trustworthiness.

The implementation of the project was designed as a three-tier web application completely separating the client-side presentation, server-side logic, and data system. The presentation layer is composed of static content delivered to the user as the front-end interface interpreted by web browsers and client-side scripting to facilitate a more dynamic user experience. User interaction with the website in the browser triggers calls to the server-side logic, which manipulates the data system and returns requested data to the presentation layer where it is dynamically interpreted and presented to the user inside the browser. The data system provides structured storage and fast retrieval of user-generated content that should be persistent.

Related Works

The core functionality of CryptoCampus draws inspiration from the popular classifieds website craigslist, where users post listings for a variety of categories in their community. On craiglist, communities are divided into subdomains by geographic areas. However, unlike CryptoCampus there is no user verification process restricting users to their subdomain

communities and listings are open to the public. The idea to make exclusive communities based on university email addresses was inspired by the original concept for the social network Facebook, which was initially restricted to users with specific email domains such as harvard.edu. Using university email addresses for verification and as usernames on CryptoCampus allows for more trust in user interactions than the unrestricted registration and pseudonymous environment of craiglist which has been criticized as potentially unsafe to use.

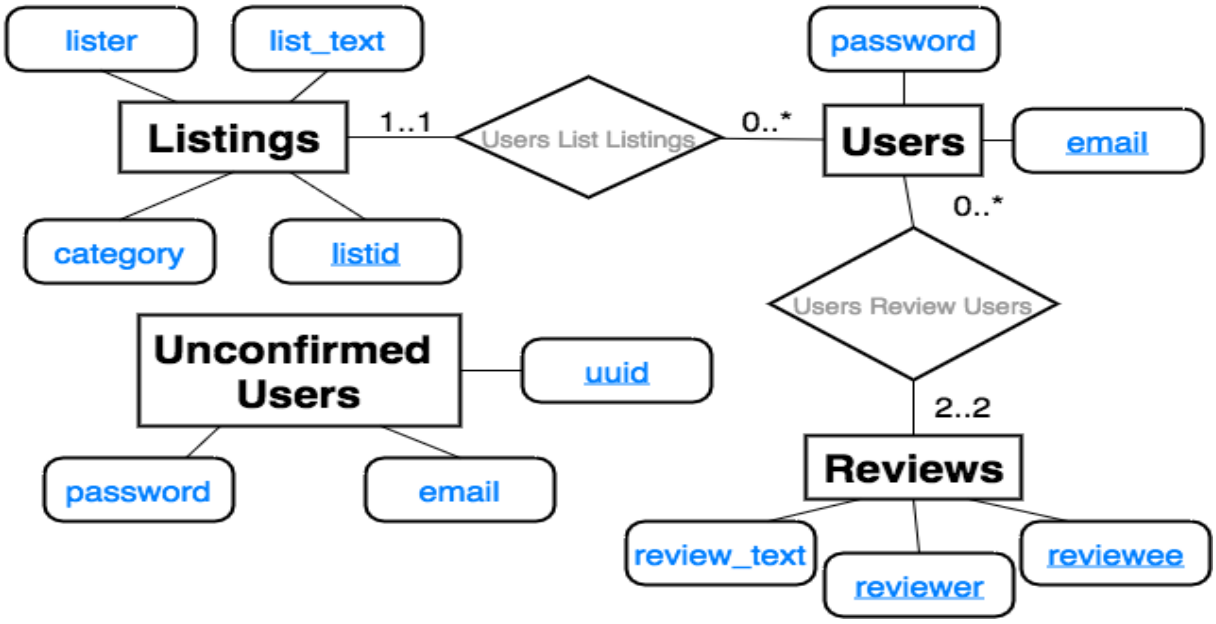
System Architectural Design

The system architecture follows the general three-tier web application design described in the introduction. The application was deployed across two Amazon EC2 instances, one containing the presentation layer and the other containing both the logic and data layers. The first instance runs Apache Web Server hosting the static web content at its root. The second instance runs Apache Tomcat server containing Java EE servlets and MySQL for the relational database storing CryptoCampus data. The DNS for http://cryptocampus.com was configured to point to the instance running Apache Web Server. The Web Server was connected to Tomcat server with mod_jk such that requests to http://cryptocampus.com/do/ are all routed to the Tomcat server worker. Each servlet is assigned to a specific path after the /do/ part of the URL depending on the requested action. Java servlets use Java Database Connectivity (JDBC) library to connect with the CryptoCampus MySQL database.

The front-end HTML, CSS, and JS was developed using the Bootstrap framework for the web page design and jQuery library for AJAX functionality. AJAX was the key to completely separating the static content presentation layer on the Apache Web Server from the dynamic content generated by the Tomcat server. Any dynamic content needed for a web page would be called for with jQuery AJAX and then inserted into the page on the client-side. Because the Web

Server and Tomcat container were hosted on different EC2 instances, enabling this functionality required allowing cross-origin resource sharing with the web domain. AJAX to the servlets was also used to authenticate the user, manage the current session, and send user generated content for insertion into the database. The functionality of each Tomcat servlet will be described in the next section, but in general the servlets interface with the database and return requested content in JSON format. It is the job of the client-side front-end to make the returned JSON data presentable. In the developed application, jQuery structures the data as styled HTML and inserts it into the page. However, any kind of application (e.g. a mobile app or a site with a different UI) could potentially be configured to work with these servlets because the JSON data is completely neutral to the presentation format. Therefore, the back-end of CryptoCampus could be an API for multiple front-end applications to use and making the presentation layer completely independent.

MySQL was chosen as the data system because of the general benefits of a Relational Database Management System for persistent storage and the ease of integration with the servlets using the JDBC library. The design and development process resulted in an entity-relationship model of the data needed for the CryptoCampus application as shown below.



The assumptions and constraints besides the cardinalities specified in the diagram were:

- Category for a listing should be non-empty and not null.
- Review text should be non-empty and not null.
- A user cannot write a review about itself.

This model was then transformed into relational tables for use in a MySQL database that could be queried and updated by the servlets called by AJAX requests from the client. The headers of the relational tables are shown below along with brief descriptions of their use in the developed application.

Relational table: unconfirmed

The unconfirmed stores the data of user accounts that have registered but have yet to confirm their account. The primary key is the randomly generated universally unique identifier (UUID) code to prevent more than one registered accounts from having the same confirmation code at the same time.

uuid varchar(100)	email varchar(100)	password varchar(100)

Relational table: users

The users table stores the data of user accounts that have been confirmed. The primary key is the user's email address, which is used as a foreign key in other tables. Note that for security passwords should not be stored as plain text, but they were for the purpose of rapid development on this application.

email varchar(100)	password varchar(100)

Relational table: listings

The listings table stores the data for listings created by users. The primary key is an automatically incremented integer. The lister attribute is a foreign key that refers to email in the users table. The list_text attribute has the text type to allow for long entries needed for a classifieds listing on CryptoCampus.

list_id int(11)	lister varchar(100)	category varchar(100)	list_text text

Relational table: reviews

The reviews table stores the data for reviews created by users. The primary key is the email address of the user writing the review (reviewer) coupled with the email address of the user the review is about (reviewee) to ensure that one user can write a maximum of one review for each person.

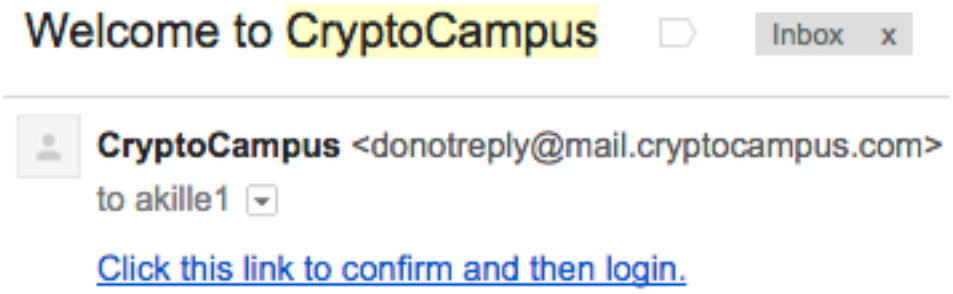
reviewer varchar(100)	reviewee varchar(100)	review_text text

Detailed Description of Components

The following is a description of each functional component of the CryptoCampus application as a servlet in the Tomcat container with a screenshot of how the developed Bootstrap front-end presents the response of the servlet if applicable. All of the servlets forward POST requests to the doGet method. All of the servlets configure the Access-Control-Allow-Origin HTTP header to enable CORS with the CryptoCampus domain that the AJAX request comes from. If a request doesn't specify a required parameter for the servlet, then the servlet will return without a response. JDBC is used for all communication with the database and prepared statements are used when handling user submitted data.

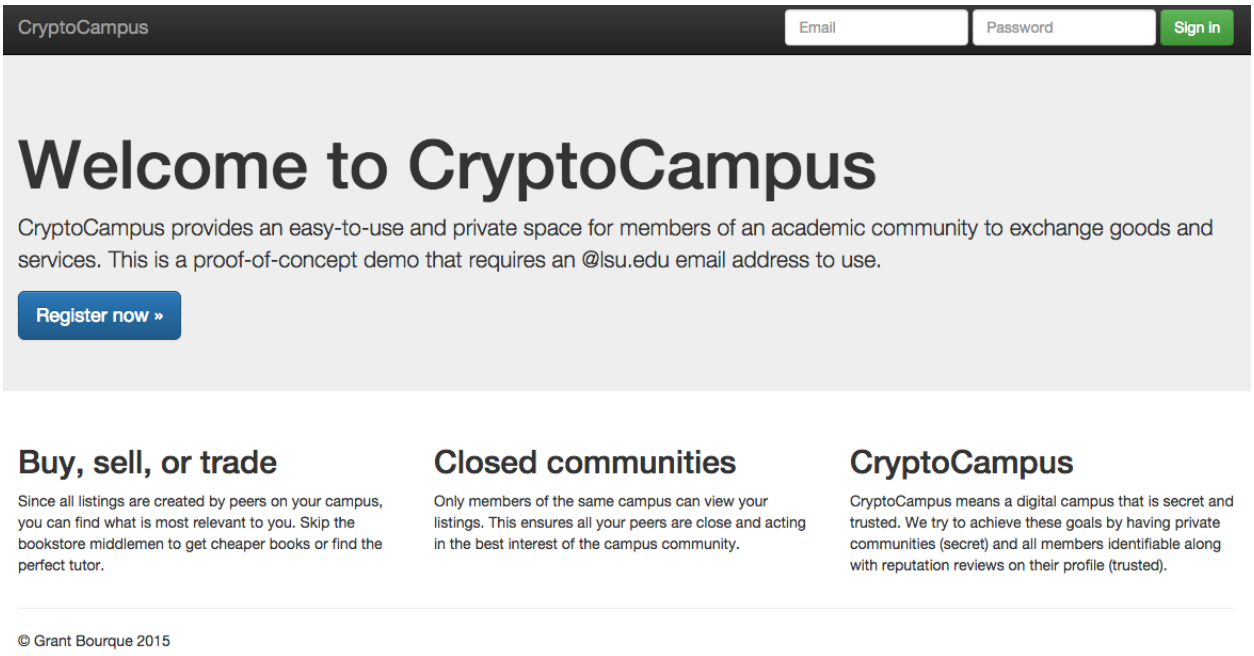
Account Registration

The account registration servlet receives an email address and password for registration of a new account. Since the developed application is a proof-of-concept demo only for LSU, the servlet checks the domain of the email address and returns early if it does not match 'lsu.edu'. Then the servlet generates a random UUID to be associated with the unconfirmed account. The servlet queries the database to check if there are any accounts (confirmed or unconfirmed) already associated with the email address, and if not then insert a row into the database with the email, password, and uuid. Then the SendGrid API is used to send an email address to the user with a confirmation link containing the generated UUID associated with the account. The following screenshot is an example of an email sent by the servlet upon registration.



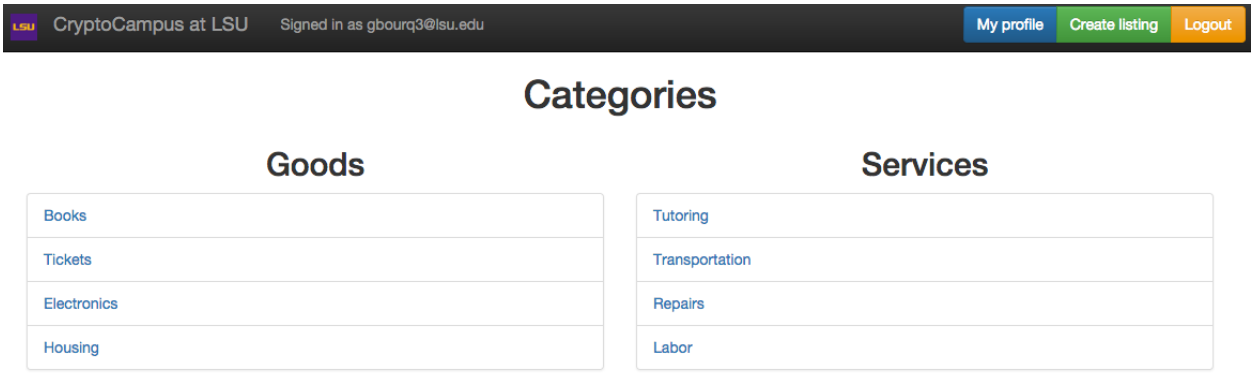
Account Confirmation

When the user clicks on the link in the email, it calls the account confirmation servlet. The servlet receives the UUID associated with the email address and checks the unconfirmed table in the database for a row with a matching UUID. If an unconfirmed account is there, then the email and password will be inserted as a new row in the users table and the unconfirmed row will be deleted. As the response, the servlet sends a redirect to the front page of http://cryptocampus.com where the user can now log in with the confirmed account.



Account Login

When the user types an email and password and clicks the "Sign in" button, jQuery will send an AJAX request to the account login servlet. The servlet receives the email and password and then checks the users table in the database for an account that has the matching email and password. If the information is correct, the servlet will create an HttpSession on the server for the client. The session information will allow the user to access the functionality in other servlets. The client will redirect to the categories page upon successful login.

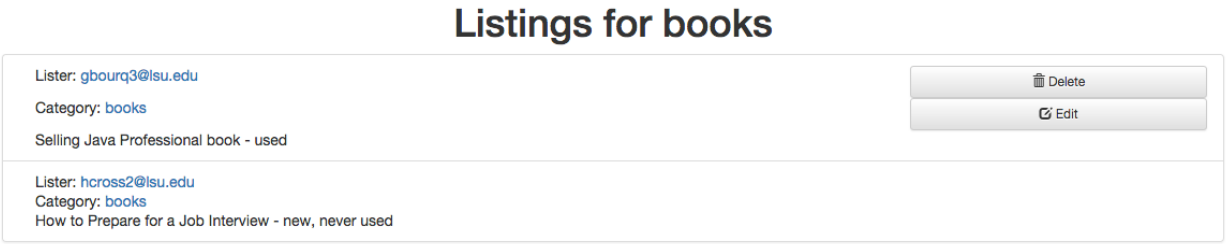


Session Information

All pages past login call a session information servlet to retrieve the server's view of the session. The servlet receives the current session ID from the request and returns a JSON object containing the email address of the currently signed user and the campus user has access to. This information is used by the client-side for presentation and verification purposes such as displaying the university's logo and a "Signed in as [email here]" message in the browser.

Category Listings

On category pages, the category listings servlet is called. The servlet receives the name of the category and then returns a JSON array of all the listings in the database under that category. The client will then use jQuery to iterate over the array of JSON and create HTML for each listing to be inserted in the page. The client uses the session information to determine if a listing belongs to the currently active user and if it does then a delete and edit button will be inserted with the listing.



Listing Creation

The listing creation servlet receives a category and list text from a jQuery AJAX call. The current email address attribute in the session on the server-side is used as the lister for the listing. These values are inserted into the database where an integer ID is assigned as the primary key for the listing.

Listing Edit

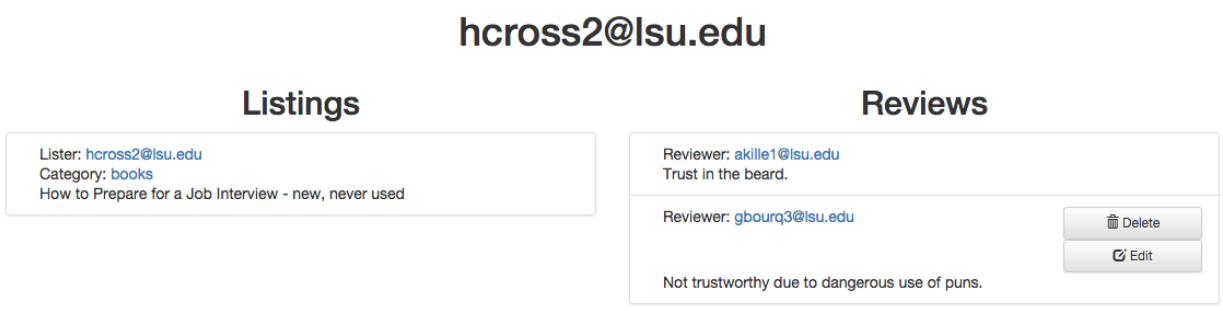
The edit listing servlet receives the list ID and list text from a jQuery AJAX call. The current email address attribute in the session on the server-side is used as the lister for the listing. The servlet uses a prepared statement to execute an update to the database changing list text of the listing with the corresponding list ID and lister. Checking the lister in the database with the server-side session information should ensure that only the authorized creator of the listing can edit the listing.

Listing Deletion

The listing deletion servlet receives the list ID from a jQuery AJAX call. The current email address attribute in the session on the server-side is used as the lister for the listing. The servlet uses a prepared statement to execute an update to the database deleting the listing with the corresponding list ID and lister. Checking the lister in the database with the server-side session information should ensure that only the authorized creator of the listing can delete the listing.

Profile Information

The profile information servlet receives the requested profile name from a jQuery AJAX call. The current email address attribute in the session on the server-side is also retrieved, and in the case the requested profile name is “me” the servlet will retrieve information using the session email address. A JSON object is returned with the profile name, a JSON array of listings from that user, and a JSON array of reviews for the user. The client sets the presentation of the data.



Review Creation

The review creation servlet receives the user the review is about (reviewee) and review text from a jQuery AJAX call. The current email address attribute in the session on the server-side is used as the reviewer for the review. These values are inserted into the database using a prepared statement.

Review Edit

The edit review servlet receives the reviewee and review text from a jQuery AJAX call. The current email address attribute in the session on the server-side is used as the reviewer for the review. The servlet uses a prepared statement to execute an update to the database changing review text of the review with the reviewee and reviewer. Checking the reviewer in the database with the server-side session information should ensure that only the authorized creator of the review can edit the review.

Review Deletion

The review deletion servlet receives the reviewee from a jQuery AJAX call. The current email address attribute in the session on the server-side is used as the reviewer for the review. The servlet uses a prepared statement to execute an update to the database deleting the review with the reviewee and reviewer. Checking the reviewer in the database with the server-side session information should ensure that only the authorized creator of the review can delete the review.

Account Logout

The logout servlet invalidates the current session of the requester which will prevent that user from accessing or creating private information in the CryptoCampus application. The client redirects to the original front page to demonstrate that the account is no longer signed in.

Conclusion

The final implementation of the CryptoCampus demo delivers a fully functional three-tier web application hosted at http://cryptocampus.com that meets the goals of providing an easy-to-use and private space for members of an academic community to exchange goods and services. The service is relatively free of bugs with no observable errors under normal usage and runs responsively and efficiently with a standard Internet speed. Because the front-end uses the screen-responsive Bootstrap framework, the website already functions as a mobile web application on smaller devices. The main area for future improvement is security, where communications should be encrypted and passwords stored securely. For the simplification of development this demo assumes that all users are in the LSU community, so expansion to include other universities would require rewriting some of the code to allow for multiple separate academic domains. Once that update is made, the full vision of CryptoCampus will be realized.