TigerList

Version 4.0

10 April, 2024

Executive Summary:

TigerList is a website for Colorado College students to buy, sell, or exchange goods and services. Mimicking an online marketplace, TigerList allows only Colorado College students to use the platform. Unlike other online marketplaces, TigerList guarantees products are within walking distance, are in safe locations and are only amongst other students. This improves safety and equity, while also encouraging good sustainability practices. Currently, Colorado College students have no single interface to message each other about buying and selling goods. Messages end up in class GroupMes, on Instagram stories, or by word of mouth, where everything typically gets drowned out, and also creates an equity concert of who gets what items. TigerList is a confluence for students to buy and sell goods without going to extreme measures.

The website first directs student's to CC's authentication platform. From there, the student can browse listings, make a listing, and edit one of their personal listings.

Problem Statement:

TigerList aims to solve Colorado College students' issue of not having one obvious domain to buy and sell goods. Students resort to class GroupMe messages or word of mouth, but those don't reach everyone and are often ignored. We aim to have goods that students want to buy or sell reach every student. Otherwise, students resort to the removal of goods in an unsustainable manner or must purchase their goods/furniture online, which creates financial concerns and raises even more unsustainable practices through shipping. By making this a student-only interface, we can guarantee a safer alternative than other online marketplaces. Students are only talking, buying, or selling with other students. We also use moderation files to prohibit the exchange of any inappropriate goods. By making our system a CC-only website, students do not have to travel far to get their goods compared to other marketplaces that cater to entire cities or multiple cities. This increases the safety of students and sustainability on our campus, as well as improves equity for students without cars. Students have a single interface that is safer and more sustainable to exchange goods.

The task of creating a functional TigerList website is multifaceted. Most obvious is the necessity of constructing a full-stack website. TigerList must be built from the ground up with both the frontend and the backend being programmed without any existing template. This creates technological challenges with features such as CAS authentication, which has presumably never been done before. This implies creating each individual feature of the user interface and writing the logic for all their functionality. Perhaps the most challenging aspect of the project is combining these two parts. This is done using a database that can store images and information and is able to read additional data and write it out. All this must happen in real-time as users interact with the website. It is these factors that make TigerList an interesting challenge in the scope of this project.

Client Requirements:

For an acceptable solution to the described problem, there are only a few hardware/software requirements but several large functionality requirements. To use TigerList, you need a device that has access to the internet. Ethernet cables could be useful for accessing the server, but are not necessary. Right now, the server is only accessible on campus and on certain wifi networks (GuestCC and gadgets-CC). In order to access the site off campus or off one of these networks, an emulator terminal like PuTTY or VWMare Horizon is necessary. Instructions for downloading these are available on CC's website. If a student does not have a personal device or computer, they can check one out at the library, which is free for all Colorado College students.

We have configured each teammate's permissions to 777 (read, write, execute) for the entire TigerList directory and all its contents. This means that our teammates can access and modify each other's directories. We have carefully designed our system to hide valuable code like API connection URLs or database IDs from our clients.

As for functionality requirements, there are several essential system requirements and a few like-to-have features. The system must have a CC authenticated login, which works for every page so that an unwanted user can not simply bypass our system with a perfect URL. By authenticating log in, we can limit privacy concerns, since the login is for CC students only. On the browse page, users need to be able to see all possible listings and click on any listings and see the details of the selected listing on a new page. The individual listing page displays all information about a product/service (price, condition, etc.) along with contact information of the user who made the posting. On the make listing page, users need to enter a title, add images if they want, and other descriptors of the goods. We also use moderation files here to prohibit a user from posting inappropriate content. We then use regex to check that all inputs are met. On the edit page, users have the listings that they created auto-populate and allow for edits. These edits also need to be saved on submit. There is also a requirement for a "my listings" page that lists all personal listings that have been uploaded that you can edit or deactivate. For the backend of the application, we must be able to query and write into a database that stores all possible listings. This connects all elements and allows users to create actual listings. These functions need to support concurrent use; we chose PostgreSQL to support multiple postings at once.

Technical Requirements:

Intended for a specific and verifiable user group, TigerList should be publicly accessible via the internet, with CAS authentication to regulate who can access it without significantly impeding student users. A valid CC email, authenticated via CC's two-factor authentication service will be prerequisite to using TigerList's services, and any device with Duo Mobile and internet access should not have trouble connecting.

In order to deploy the site in a scalable, flexible, and free way, TigerList requires a remote server through which this authentication will be administered. The server must go through port 80 to ensure a simple URL, but then sent back to listen on port 3000 to stay comptable with Next.js requirements.

Responsible for storing and providing access to information, TigerList requires a relational database that allows the site to quickly retrieve other user's listings based on search criteria, and to post and see new listings in real time. The database should be set up so that listings can be safely edited without affecting reference information or any unrelated listings. We need a query language that can support concurrent postings. The database must be set up and self-hosted in the remote TigerList server to successfully query.

TigerList needs a secure server-side script to communicate with the client side to support the entire backend of our application. An API is necessary for the security of this communication. This code needs to be compatible with Next.js new routing system and naming conventions.

The last security functionality we need is a moderator functionality. This includes having a set list of moderators who have the highest authority of what content can and cannot go on TigerList. Our app needs to check user emails as they login and use that to conditionally render some pages. Moderators get 2 extra pages: a flagged listings page and a flagged listing page. The flagged listing page shows all posts that have been flagged. From there the moderated users can delete or accept any flagged listing. The single flagged listing page happens if a moderated user wants to click on a flagged post to see more. From there they can also accept or delete the post.

System Design:

TigerList's programming logic is divided amongst individual pages that function differently across the website. Our codebase has an 'app' directory, which has subdirectories. These subdirectories represent each individual page. Each directory contains a 'page.js' script which determines that particular page's visual components and functionality.

Much of the user interface is composed of smaller components that are more easily organized on a web page. The component files contain the logic for features such as buttons, links, drop-down menus, and text bar. These components live in a 'Components' directory in the root directory of our project. The 'page.js' files then import these components, resulting in a scalable, flexible, and understandable system.

Our API lives in the 'app' folder as well. It handles the communication between the server-side functionality and the client-side operations. Within the API folder, there are subfolders for each operation that can be requested of the server-side, and a 'route.js file' within each of those. Each API route manages one or multiple RESTful API request types, such as 'GET' 'POST' and 'DELETE.' This enables the client side to interact with the database following established conventions. Most API routes use a common set of methods defined in api/dbTools to establish a database connection, query the database, close the connection, and take actions based on the success or failure of the query. In the name of security on the client side, we use environment variables to conceal the API URLs.

In our root directory, we have a 'server' folder which contains one script: 'cas.js'. This file creates a cas handler so that we can authenticate users with a Colorado College email. Upon build time, the application uses the server.js script (which is not in the 'server' folder, but in the root directory) to send the users to Colorado College's CAS portal.

Lastly, we have a number of accessory text files and bash scripts. The text files contain instructions on application setup (README), and SSH instructions. These live in our root directory. The bash scripts live in our server in the 'j_moran' folder, and automate various server processes.

Languages:

The TigerList system is programmed primarily using JavaScript. This is done through the Next.js React framework which incorporates pieces of both HTML and JavaScript for formatting and style improvements. We also use bash scripting when necessary for tasks like application set up, permission configuration, redirecting firewall traffic to appropriate ports, pulling from GitHub, and toggling authentication on and off.

Next.js Framework:

The TigerList website is constructed largely with the help of Next.js. This is a framework that uses the React JavaScript library to aid the construction of a web application. Next.js does not construct a website for clients with the information they want to include. It allows clients to configure a website they envision and provides certain tools to simplify this process. The user interface for our web application requires a significant amount of programming with JavaScript, HTML, and CSS. Next.js has allowed for optimizing this web application through its layout, formatting, and component features. In terms of the backend, Next.js ensures security with server side scripting; by default, all components are rendered server side. Users then can use a simple keyword to trigger the use of client side components so as to easily differentiate from the two.

Database:

After exploring a few services, we settled on using a self-hosted database with PostgreSQL. A local project database provides us with a more flexible design process and eliminates the risk of rampant expenses with third party providers. The database can be manipulated with the dedicated 'psql' shell, local scripts for testing and maintenance, and of course by the API methods described above. We initially created and configured the database using shell commands, and it now runs on the server independent of our project, with no need to recompile or restart like the rest of our code.

Data for Tigerlist is stored in two tables within our database: 'usertable' and 'posttable.' The user table stores information associated with each user, with the primary key being their unique Colorado College email, and booleans to indicate their moderator and banned status. The post table uses a unique index value as the primary key for each listing, and stores information displayed to users, including the price, description, and category of the post. The post table also stores important information including the time a post was made and its flagged and hidden status, so the API can determine when to show certain posts and how to sort them. It is important that the post table stores a user's email for each post, ensuring accountability and enabling requests for all posts made by a given user.

Email moderation:

We set up an email service to ensure moderators get real-time updates about TigerList's content. This enables moderators to keep tabs on posted content while away from the server and website. An SMTP, or standard mail transfer protocol, allows the automated sending of emails from a server like ours. Methods imported from *app/moderation/* allow the API to prompt sending emails, passing off the relevant content to be formatted and delivered to moderators. Email notifications are currently set up to be sent whenever a listing is added, edited or flagged.

Authentication:
Our authentication system is CAS. CAS has a number of supported libraries for most languages; we chose a GitHub repository which is a near complete implementation of CAS Client middleware for Node.js. After adding this logic to a cas.js script which is sent to sever.js script upon build time, users are redirected to Colorado College's authentication portal. Cas.js contains the server side CAS authentication logic using the 'https-cas-client module, and is then integrated into server.js with a casHandler object using a require statement. A valid email triggers a yes on a flag variable which sends a notification to a user's device; if the user follows the correct prompt on their phone, then they are redirected to the application's browsing page.

Installation Procedure:
If a user is on campus and connected to guest or gadgets wifi, then no special software installation is required for use; all that is necessary is navigating to "http://tigerlist.coloradocollege.edu" in their preferred browser. In order to view the site off-campus, the user needs to use CC's desktop virtualization software VMWare Horizon Client. Instructions to download are linked here: https://www.coloradocollege.edu/offices/its/guides/connect-from-off-campus.html.

When downloading the VMWare software to whichever operating system, it is absolutely essential the user downloads no later than version 2909! The TigerList server is not compatible with the updated versions. If at any point the user is prompted to upgrade versions, they must select no. Once VMWare is installed, the user is presented with a remote browser, and then they can enter the same URL in that machine's browser. Unfortunately, so long that our code base exists on ITS server, there is no way to access the server off-campus without VMWare.

Operation Procedures:

For one of our team members to launch TigerList, the first step is SSHing into the external server for TigerList provided to us by CC. The command is "username@hostname", so, your_cc_email@tigerlist.coloradocollege.edu. Currently, for this to work, you need to be on campus and on guest wifi, or in an emulator terminal. Once SSH'd in, we have provided shell scripts in the remote server that automate all relevant procedures. 'setup.sh' builds and deploys TigerList to our custom url. 'cas-on.sh' and 'cas-off.sh' toggle authentication on and off, respectively. 'reboot.sh' redirects TCP traffic to the appropriate ports, which must be executed after any sort of server reboot or failure. Lastly, 'update.sh' pulls the most recent version of our code from GitHub, to help maintain version control in the remote server. There are a number of txt files which contain instructions and other commands.

Moderator access once TigerList is running is controlled manually in the back end. There is a list of authorized users that can only be edited by those with access to the code base. When any TigerList user logs in their email is saved from CAS authentication and checked against the list of approved moderators. If their email is on the list, then they will have the ability to use a hidden moderation page within the website to approve or delete flagged listings. Moderated users will also receive email notifications whenever a listing is flagged to ensure a timely response and a safe website.

User Procedures:

After the system's owners successfully run the Next.js application in the remote server, all a client needs to do is enter the URL "http://tigerlist.coloradocollege.edu" in their browser, or in VMWare Horizon Client if off campus. Refer to installation procedures if this is the case.

Once authenticated, the user is then brought to the browse page, where they can see all listings currently stored in the database. From here, they can browse through listings, and click on a listing if they are interested. This will display a page with the product information and the seller's contact information. From there, the user can contact the seller on their own. We decided not to implement any sort of transactions within our application since students prefer to work out a payment method on their own, and to avoid security concerns with exchanging financial information.

There is a consistent navigation bar on each page to get to any other page - make listing, search, or a profile option to edit your own listings. We provide users with a simple interface and options to consistently navigate to any other page in the system. In order to sell a good, the user should click the "Make Listing" section of the navigation bar. This takes the user to a page which prompts them for product name, price, etc, all which contain regex logic to verify inputs. The user can also add an image if they wish. Once finished, they click the "Make Listing" button, which will result in a small toast verifying their listing has successfully been uploaded. The listing is now on the browse page for all to see.

To edit a listing, the user must click the profile icon, which is the icon furthest to the right of the navigation bar. This will display a page with all of that user's listing. From here, the user can click "edit" on the listing they wish to edit, which will bring them to a page similar to the "make listing" page. Users do not have to sign out of the app, as it is integrated with Colorado College's single sign-on protocol and they will be signed off automatically after ten minutes of inactivity.