Abstract

Backchannel communication is the practice of having a secondary electronic conversation while a primary spoken conversation takes place. Backchannel has been applied to different domains and settings and one of the most significant applications of backchannel is education. Backchannel allows one to establish a more student-driven classroom and encourage students to ask more questions without having to interrupt the instructor. Backchannel has been applied to enhance the classroom experience in different places such as Harvard University and Cornell University. However, the platforms used are outdated and do not provide the user experience necessary to sustain a real-time conversation.

1 Introduction

This project seeks to explore the intersection between technology and education, especially the application of backchannel in the classroom. Students may shy away from asking questions for a plethora of reasons including potential for class interruption and fear of asking a question that may seem unintelligent to their peers, especially in bigger lectures. For these reasons, the main motivation of this project is to encourage students to ask more questions during lecture and provide a classroom experience that is equally led by the instructor as well as the students to maximize learning.

As a matter of fact, the intersection between technology and education has produced several tools that help enhance the classroom experience. For example, Piazza is used on a daily basis by thousands of students and professors in over 30 universities across the country. It has even extended to becoming a way of tracking class participation. In spite of Piazza proving to be a useful tool in education, it is meant to be a discussion tool post-class and not during class. This project aims to fill this gap.

This project is designed to be a Q&A tool that applies the backchannel model to allow students to ask and vote on questions anonymously during lecture and engage in the learning experience more directly. Furthermore, it is designed with the instructor in mind, giving the instructor access to valuable data that can be used to extract student feedback. At the end of the class and at the press of a button, the instructor is able to archive all of the questions and send automated emails to all of the class participants. This approach makes it easy to reference questions in the future and to determine which parts of the course were harder for students to understand. This allows for a feedback portal which can be
used by the instructor and future instructors of the course to improve the class structure and potentially analyze this data to gain more insight.

This project was built with modularity and scalability in mind. The project is well structured and code is split among files to make it easier for future developers to understand the code and add to it. Moreover, clear documentation was written to avoid confusion and to easily understand how to install and run the project.

Similarly, user experience is a crucial aspect of any real-time project. This project employs websockets to propagate any changes made by a network request not just to the client who issued the network request but also to all other clients connected to the same course session.

2 Project Description

This project was built using cutting-edge technologies. The backend was built using Node.js and Express. Express is a Node.js based web application framework that provides HTTP functionality which can be used to build robust APIs. The frontend was built using React, a highly effective frontend Javascript-based library, and Material-UI which provides plenty of styling options, allowing us to build a seamless user interface.

Built with scalability in mind, the project structure was designed to allow future developers to easily familiarize themselves with the code integrate themselves into the project. The frontend and the backend were both separated and divided into several folders and files to make the functionality and intention of each file clear for the person trying to understand the project’s source code.

This project was designed for two different types of users, students and instructors. Therefore, the functionality of the application differs based on the type of user. For this reason, the user interface was designed to be role-based, showing different frontend components based on whether the user is a student or a professor.

The purpose of this project is to bridge the gap between student and instructor and to allow the students to have more of a primary role in directing the lecture. However, in order to efficiently achieve that, the communication tunnel between the students and instructor should be in real-time and have strictly limited lagging or delayed effects. Thus, I decided to implement websockets in this project using the Socket.io library. Websockets were used to emit and listen on events that allow the students and instructor to see new questions posted and voting status on existing questions in real-time without having to press the “refresh” button. This functionality maximizes user experience to allow for real-time interactions between the students and instructor without any unnecessary time lag.

Given that all questions are anonymous, I had to implement a profanity filter to make sure that inappropriate expressions did not pass through and were not posted for everyone to see. The profanity filter was built using a regular expression that encompasses the majority of profanity words including slang and urban expressions.
3 Database

3.1 Technology

I decided to use MongoDB for several reasons. Firstly, NoSQL non-relational databases are easier to manage and have a higher level of flexibility than SQL databases. Furthermore, NoSQL databases support Map Reduce which makes NoSQL more scalable. Secondly, MongoDB was designed for high reads and writes. Given that this project revolves around providing a real-time experience and utilizes websockets, MongoDB was the best fit because of its higher performance in database reads and writes. The flux of reads propagated through all the clients when a question was asked or voted on can require some computational power that MongoDB was able to minimize. Finally, there is a strong community for the MERN stack (MongoDB, Express, React, and Node.js).

3.2 Design

The database schema was straightforward because there were not many distinct entities required and there were not any complex relationships. There were three main entities: courses, questions, and users. To avoid complicated database queries and inefficiency, the question entity was designed to be a subdocument of the course entity. Each course has a password that is automatically generated as a 5-letter alphanumeric string.

4 Backend

4.1 Design

This project was built with scalability and user experience in mind. For this reason, the project is split into backend and frontend, both of which communicate through a REST API. The backend was built using Nodejs because of its non-block I/O model which deliver a substantially higher performance than threaded server environments. As such, we used Express as the backend framework, also known as the "de facto standard server
framework for Node.js", due to it being lightweight and flexible. In order to communicate with the database, we used Mongoose, the official ORM for MongoDB, which allows the execution of clear and succinct database queries to work with data in a smooth manner. In order to optimize user experience, I chose to implement websockets in the backend to allow for a seamless experience; when students voted on or asked questions, the goal was for them to see these changes on the screen without having to click the “refresh” button to fetch the new/edited data.

4.2 Development

The development of the backend consisted of three different tasks: (a) creating the models and translating the database schema, (b) building the routing logic to enable the frontend to communicate with the backend in an efficient and secure manner, and (c) ensuring that all keys used were hidden (e.g. the database instance on mLab) and could not be found on the Github repository where the code was hosted.

There were three main models that represented courses, questions, and users. However, instead of having the course and question models be separate, I decided to implement questions as a subdocument in the course. This design decision was made because questions were heavily tied with courses and therefore, it made dealing with courses and course questions much smoother.

To implement websockets, the Socket.io library was used. A sockets service was created and imported in the server. This service had several event listeners (e.g. listen for a question submission event). These event listeners would send data back to the client whenever one of these events was activated. This data was used to update the frontend right after one of these events were emitted, which led to a seamless user experience and allowed students to better engage with the classroom and each other’s questions.

5 Client

5.1 Design

Given the importance of user experience in this project, the client side of the project was built with minimized lag in mind. The goal was to provide a seamless experience when the students and the instructor were interacting with each other. I chose React because it has high performance. React creates a virtual DOM, a tree based on JSON created by React, which makes re-rendering efficient.

React moves data around its components using states. However, moving data from one component to another can result in redundant code, especially if there are several layers of components. For this reason, I decided to use Redux, which wraps all the components with a "Redux Store" which is a global state that all components can get access to. This results in more modular, more scalable, and overall cleaner code in the frontend.

In a similar fashion as the backend, the frontend uses websockets to emit and listen to events and consequently updates the components according to changes in the component’s state or the global state provided by Redux.
5.2 Development

In order to maintain modularity and readability, the frontend code is divided into three main parts: components, actions, and reducers. Components are split into two types: smart components and representational components. The purpose of the smart components is to fetch and write data while representational components, also known as "dumb components", are used to display the views to the users. Actions contain all the files that are responsible for creating AJAX requests to the Express API. Reducers are responsible for updating the global state whenever an action is dispatched.

Figure 1: The landing page from the instructor's view

Figure 2: The landing page from the student's view
6 Application Architecture

Provided that the backend (Express API) and frontend (React client) are separated, the application architecture determines how these two components (with all of their different parts) interact with each other and is therefore crucial to the functionality and performance of the application. In this project, I used two distinct servers, one for the backend and one for the frontend. The backend server’s sole purpose is to serve data from the database. The frontend server’s purpose is to take all of the JavaScript components that build the user interface and bundle them together into a single bundle.js file. To allow for routing to go to the proper server in the development environment, I set up a client on the client side that would redirect requests that were seeking to fetch or manipulate data to the express API; otherwise, it would go to the react router.
6.1 Security

The motivation to use this architecture stems from the accompanying security benefits. Cookies are the main anchor when it comes to authentication. Once the user goes through the OAuth flow, the backend populates the user’s cookie with user-related information, which is used to identify the user when sending network requests. In the development environment, the backend server is set on localhost:5000 and the frontend server is set on localhost:3000. There are two scenarios in which security issues may arise.

First, if the browser is at localhost:3000 and issues an AJAX request to the server at localhost:3000 then cookies will be included in the requests, which is the desired behavior. However, if the browser is at localhost:3000 and issues an AJAX request to a server at localhost:5000 (i.e. a server on a different domain or port) then cookies by default are not included. This is because the browser assumes that if it makes a request to a server other than the one that it’s currently visiting, then that could be an indication of some malicious JavaScript code that is attempting to make a malicious request to a different domain. This security concern is avoided in our architecture because requests are always directed to localhost:3000 and the proxy uses its rules and settings to determine whether to direct the network request to the backend server at localhost:5000. In that case, the network request will have the cookie information since the browser does not know that the proxy exists. Second, sending a network request to a different domain or port is considered to be a CORS request (Cross-Origin Resource Sharing) which is considered unsafe by the browser for similar aforementioned reasons. While a server can be configured to deal with CORS requests and achieve the desired flow, adding the proxy on the frontend resolves this issue.

In addition to the security benefits, this application architecture works in tandem with the authentication flow.

Conclusion

To allow for my project, “QME”, to be usable, I deployed it to the web using Heroku. For security concerns, I set up environment variables on Heroku that store sensitive information such as the key to the deployment database.

The intersection between education and technology is very fascinating with plenty of room for further exploration and research. QME is a great example as how to enhance the classroom experience using technology.

On April 25th, 2018, QME was tested out for the first time in CPSC 365, a computer science class at Yale University. Students actively engaged in asking questions during a class review session and the results were promising. As this project develops further, I expect to transform this project into a Q&A platform that couples as a data hub to gain further insight on how to optimize the classroom and lecture experience.
Figure 6: A demonstration of QME used in CPSC 365

<table>
<thead>
<tr>
<th>Questions</th>
<th>Answers</th>
</tr>
</thead>
<tbody>
<tr>
<td>When will HW2 grades be out?</td>
<td>14.41</td>
</tr>
<tr>
<td>When will HW4 grades be out?</td>
<td>14.41</td>
</tr>
<tr>
<td>Can we go through an example of reduction</td>
<td>4.41</td>
</tr>
</tbody>
</table>