CS65 Peerist Problem Statement

Jorge Manzo
Darius Moseley
Christopher Hoskins
Michael Chan

CS 461
Fall 2019

Abstract

Academic writings require a lot of time investment when going through feedback and revisions. Currently, there aren't any available tools to allow academic writers to receive feedback in a timely and unobtrusive manner. There is also a need for keeping a history throughout the life of an academic paper. Peerist will provide a service to allow academic writers to be given feedback on segments of writing to then be composed into larger papers. This document will describe the background and justify the need for Peerist as a service to academic writers. We will also describe how Peerist will address a large concern of academic writers that want feedback on their work: plagiarism. The requirements by the client are discussed as well the client-specified technology stack of React.js, Next.js, ZEIT Now, Hasura, GraphQL and PostgreSQL. The client's performance metrics for the proof of concept, and how we can produce it, are also discussed. The implementations that we can expect to be provided by the client are also included in this document.

Definition and Description

The aim of Peerist is to create a tool for collaboration in the space of academic research papers. There is a lack of tools available for researchers to collaborate and peer-review their work. There are document editing tools currently available, but they lack the specific tools and features required for academic papers. Currently the process of providing and sending feedback on academic papers is time-consuming. Peerist will provide an online platform to make it easier for users to provide feedback for academic works in a way that is modern and pleasant. This service will also enforce private peer-reviewing once segments are composed into larger components.

Users should be able to upload segments of their academic writing. When edited, these segments should be individually versioned based on the changes provided by editors. This is similar to how version control systems such as Git keep track of changes to code. The process begins with segments of the writer's work, as this makes it easier for the reviewers to provide higher quality feedback. Because these are segments of the writer's work, there is little incentive for plagiarism.

When a user is ready for feedback, the user can request to be matched with editors who will provide it. These editors are picked from the public of other users on Peerist. The editors will be given prompts to help guide the editing. Changes made here should be kept track of. The user should be able to see a history of the edits and changes made to their segment of writing, and see which user contributed which edit. There will be a limit to how large these text segments can be. Users will be able to upload as many portions of their writing as desired. When ready, the user will be able to compose their segments into a larger product, called a Paper.

A paper's history will still be kept and versioned like segments. When requested, the user can also have their paper shared to editors for feedback. However, to prevent plagiarism these full papers will not be shared with the public user-base of Peerist. Instead the writer shall specify editors from a circle of trusted individuals to review their work. Here the feedback will be provided just like it is provided for segments, but for the paper as a whole.

Proposed solution and Technology

Our implementation will be built in a sandboxed environment. We will be using Next.js, which makes it easier to write React JSX. This package takes the setup time and configuration out of building an app so we can just focus on the code. React is a JavaScript component-based framework for building web applications. It is coded in JSX (a flavor of JavaScript). One of the benefits of using Next.js is it allows us to use server-side rendering. This makes loading content in the application much faster, as the server streamlines the content sent to the client.

Our application will be hosted on a service platform called ZEIT Now ("Now" for short). Now is a serverless deployment service. Because Now is serverless, it handles server management, configuration, scaling, and monitoring. This means that all we have to implement is what code should be called upon an HTTP request. Their service can compile static site code such as React's JSX into browser-ready code, including server-side rendered code. JSX is not understood by browsers, so the components we make need to be "compiled" into JavaScript, HTML, and CSS.

Now also provides computing service for lambdas, which are snippets of code functions to execute on their servers. Now's pricing for this computing service scales up with the usage of the application, which keeps the cost of the service low. This code is called whenever an event such as an HTTP request occurs, and our client has suggested we write these in TypeScript. We will be creating these functions to communicate with Hasura.

When needed, our lambda functions will communicate with Hasura as the API gateway to our PostgreSQL database. For our development, we will setup docker to run Hasura and PostgreSQL locally. We will use GraphQL queries when interacting with Hasura and the PostgreSQL. We can request specific data, down as many layers as desired, and get back only what we requested.

Performance Metric

Our client has requested we implement a working proof of concept of this project. Our development for the most part will be front-end focused with some back-end work when needed. Because we are using React, our implementation should be easy to be installed into their existing codebase with minimal re-implementation. We are not required to integrate our components with their existing codebase.

There are also portions of the project we are not responsible for implementing. One of these is the match-making of writers to reviewers. Our client has stated that they are implementing their own algorithm to pair writers to the appropriate users. Our client will also be providing some of the Hasura setup.