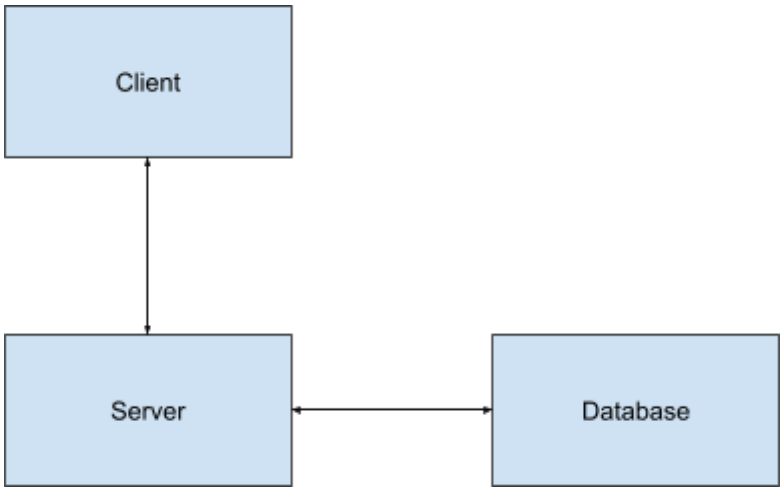
Student Number: 10585625
Youtube Link: https://youtu.be/qpzfoUxp3Pk
Github Link: https://github.com/dampo4/Pictionary-website

Requirements

The application is aimed at young children as a form of communication and collaborative work. The project was originally intended to be a pictionary game however instancing groups of people to play on their own proved quite difficult so the scope was revised. There are several features included which make use of a database and websockets. Clients are able to draw on their canvas, this is then converted to image data and sent to all other clients where it is redrawn on the community canvas. Users must make an account in order to use the chat functionality which makes use of a database. Clients can input a username and password and, if the username is not already in use, create an account. They can then login with those credentials and send messages. These features seemed more apt than an extensive database system as I am a games development student.

Design



The System architecture is very basic. The client has no direct interaction with the database and any requests are first passed through the server. The client's own canvas is updated on the client's side however the community canvas updates are sent to the server then distributed to all connected clients. Account creation requests are sent to the server which then queries the database and performs the relevant action based on the database's response.

Testing

Some initial unit testing was done on manipulation of the mongo database using Mocha however, due to being unfamiliar with Mocha, way too much time was spent debugging the tests themselves; more time than Mocha would have saved on a project this size. Minor user testing was done by a friend from my games development course. The testing's key goal was to identify any bugs which arose during development. One of the bugs identified through this was the fact that the community canvas would get wiped every time a different client added to it which was eventually fixed. Another bug is that, when the browser size is made smaller, the chat window gets hidden under the canvas. A change was made so that this no longer happened however that led to another bug. The canvas could no longer accurately track the position of the mouse so the change was reverted and this bug still exists.

DevOps pipeline

The development environment consisted of Atom (IDE) and a combination of Github and Github Desktop. In addition to this, nodemon was used to constantly reload the server every time a file changed as well as mocha for unit testing. Every time a push was made to Github, the site was in a working state with no known bugs and a large element had been added. Because of this, continuous integration was not implemented ideally (smaller changes to code should have been pushed more frequently rather than large chunks all at once).

Personal Reflection

In terms of technologies used, they mostly worked seamlessly. The only problem I encountered in this regard was with Atom. At one point I was simply trying to add a script tag in my html script and this was causing the rest of the file to be unreadable. After a long time googling for answers without success, I closed down the file. Upon reopening it, everything now worked perfectly fine. This slight hiccup would not deter me from using Atom again in the future, though, as it is very user friendly in terms of colour scheme, autocompletion of code etc.

In terms of my own coding practices, there are a few things I would change. Firstly, my knowledge of coding structures is poor so I would brush up on that if I were to undertake the project again. Secondly, I find it hard to drag myself away from coding in order to make continuous pushes to Github (which I know is bad practice). To rectify this in the future, I may set an alarm to go off at frequent intervals to remind me to push my code.