# Description

This is simple chat app powered by Node.js, Express, Socket.io, and Redis. It uses a web-interface, via which users can join the chat using a username, send and receive messages to and from all the users currently in the chat room, and see a brief chat history.

# Instructions

To run the app locally you’ll need the Node Package Manager (NPM), Node.js, and Redis server installed globally.

To start, navigate to project directory (“Learnosity Chat App”), open the console and install (npm install) the modules required for the server to run. Once that is done, enter “node app”, and if everything went well you should see a “Listening on port 1000” message, which indicates that the server is running.

To open the chat app, open a tab in your favourite browser, and navigate to <http://localhost:1000>. There you will be prompted for a username. Enter the username, and press submit. The chat window, and the user list will then open.

Once you join the chat type in a few messages, and then open a new browser tag, and repeat the process above again. Join as a different user, and check if the past messages are displayed.

# Reasons for Approach

Ever since I’ve started learning Node.js I’ve wanted to create a responsive real-time application, such as a chat. This Tech Test gave me a perfect opportunity to both learn something new, and to showcase my knowledge in full-stack Javascript.

I chose Node.js as the server of the app because it’s the back-end platform that I am the most confident on, and used Express because it’s the premier time-saving module for Node.js.

I selected Socket.io because it fits incredibly well for real-time, multi-user applications.

I picked Redis as the storage module because its fast (perfect for a real-time chat app), its platform independent (can easily be scaled cross-platform, cross-application, etc.), and it’s a technology that I haven’t used before, and was eager to learn.

# Future Enhancements

The development time of the application had a strict time constraint, which left a lot of features unimplemented. Here’s a list of the things that I might do in the future:

* Output a client joining/leaving the chat
* Separe storage into a separate module
* Use a templating engine to serve html
* Implement unit tests
* Implement chat rooms and private messaging
* Auto scroll chat if the height of messages is larger than container
* Style the scroll bar
* Implement keyboard shortcuts
* Implement customizability per user
* Refactor the front-end to use Backbone.js