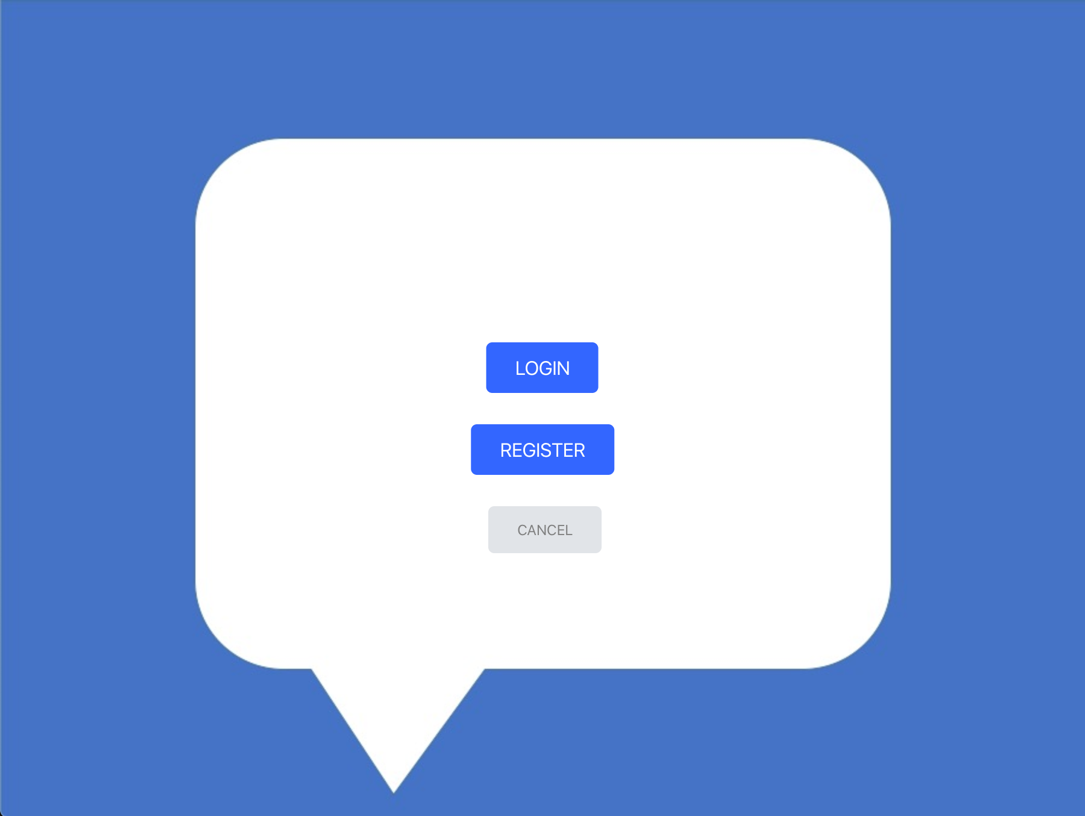


Project Engineering

Messaging Application

“Chapp”

Daniel Fitzgerald



Bachelor of Software & Electronic Engineering

Galway-Mayo Institute of Technology

2019/2020

Declaration

I hereby declare that this project and all associated work is that of my own, except whereby otherwise stated and accredited. Any work included or incorporated into this project is acknowledged and referenced.

This project has been undertaken and presented as part of the requirement for the degree of Bachelor of Engineering in Software & Electronic Engineering at Galway-Mayo Institute of Technology.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Daniel Fitzgerald

Table of Contents

[2 Summary 4](#_Toc39835579)

[3 Introduction 5](#_Toc39835580)

[4 Full Stack Web Development 5](#_Toc39835581)

[5 Project Architecture 6](#_Toc39835582)

[5.1 Architectural Diagram 6](#_Toc39835583)

[5.2 Development Tools and Platforms 7](#_Toc39835584)

[6 Database 8](#_Toc39835585)

[6.1 Apache Cassandra 8](#_Toc39835586)

[6.2 Docker 8](#_Toc39835587)

[6.3 Installation and Configuration 9](#_Toc39835588)

[6.4 Keyspace and Table Setup 9](#_Toc39835589)

[7 Server 12](#_Toc39835590)

[7.1 Apache Tomcat 12](#_Toc39835591)

[7.2 Tomcat – Cassandra Connection 12](#_Toc39835592)

[7.3 WebServlets 13](#_Toc39835593)

[8 Client 16](#_Toc39835594)

[8.1 React 16](#_Toc39835595)

[8.2 JSX and Virtual DOM 16](#_Toc39835596)

[8.3 Components 17](#_Toc39835597)

[9 Network 21](#_Toc39835598)

[9.1 Server and Client 21](#_Toc39835599)

[9.2 JSON 22](#_Toc39835600)

[10 System Integration 23](#_Toc39835601)

[11 Conclusion 24](#_Toc39835602)

[12 References 25](#_Toc39835603)

# Summary

The aim of this project was to design and develop a full-stack messaging application. The application consists of multiple elements to function:

1. Database(DB) – storage of application data.
2. Server – performs reads/writes to the database, handles requests from the client and serves the client with data that is requested.
3. Client – renders a Graphical User Interface (GUI) and sends HTTP requests to the server.

The application allows two users to communicate with each other in a one-to-one conversation context via messages in text format. The user inputs a message which is sent via HTTP request to the server. The message is then written as a text string to the DB. This message and all other messages are stored in the “conversation” between the two users. The content of the conversation is rendered to the GUI so the user can view all messages that have been posted to the conversation.

For this project, Apache Cassandra – a NoSQL database – is the software being used to store all necessary data. This data is stored in three tables that store a user’s personal information, user’s conversation list and the message entries for a conversation. The server component is deployed in an Apache Tomcat webserver (written in Java) and uses HTTP web servlets to process incoming requests. The final component is the client and is designed using React. This is a JavaScript library used particularly for UI design.

Various additional tools and software were used to organize, develop and test the project’s functionality. Some of these were Maven, Git and Postman. Maven was incorporated into the server component of the project to manage dependencies needed for the application. Git and GitHub was used at all stages of development to create and maintain a remote master code base. Postman served as a way of testing the API of the server’s web servlets.

At the conclusion of the project, the application can successfully be deployed and run on separate desktop machines, allowing two-way networked communication between users.

# Introduction

The motivation for this project came from the importance and essentiality of instant connection today. In the present age, it is our society’s state of connection in all things that enables our industry, commerce and personal lives to function. In the business world, instant and efficient delivery of information is now both a key to success and also one of its requirements. Communication with the wider world has become a near-necessity in our personal lives. The success of social media platforms and messaging applications testify to this.

The scope of this project was to deliver a simple messaging application capable of allowing two users to communicate with each other in a one-to-one messaging/chat environment. The user data of the application is to be persistent and accessed and written to by a server. Client-Server components should be able to exchange HTTP requests/responses to update the database and user interface.

# Full Stack Web Development

A major attraction to undertake this project was the opportunity to develop a full stack application - gaining knowledge and experience in both server and client software development, in addition to improving database understanding.

The technologies chosen for this project stack are Apache Cassandra – Tomcat - React.

# Project Architecture

## Architectural Diagram

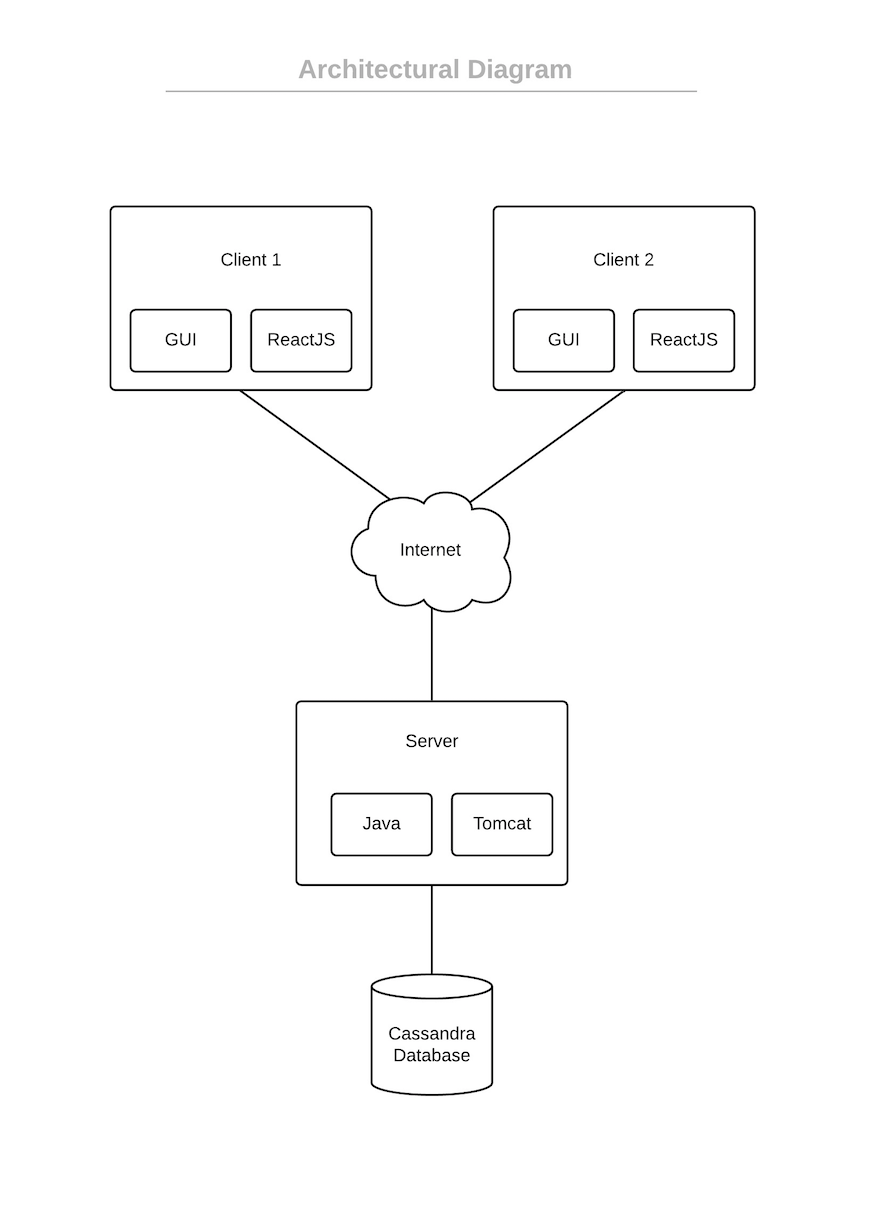


Figure 5‑1

## Development Tools and Platforms

**Database**

* *Apache Cassandra:* this NoSQL database was deployed locally.
* *Docker:* used to deliver the Cassandra database instance as a virtualized software package known as a container.
* *Kitematic:* a GUI application used to interact with and organize Docker containers.
* *CQLSH:* command line shell used to manually query the Cassandra DB using Cassandra Query Language (CQL).

**Server**

* *Language:* All server functionality developed in Java.
* *Apache Tomcat:* Java HTTP web server environment. Web servlets are responsible for processing client requests and DB reads/writes.
* *Maven:* Used for project management, adding needed dependencies for the project e.g. JARs required to configure Cassandra DB.
* *IDE:* IntelliJ Ultimate Edition.
* *Postman:* An API-testing application tool. Used to send HTTP requests to the Tomcat web servlets and analyze responses from the server.

**Client**

* *Languages:* JavaScript, React JSX, HTML, CSS
* *React*: JavaScript library used to create User Interface, process user input and then send HTTP requests to the server.
* *IDE:* Atom
* *Debugger:*  Chrome Web Development Tools

***Network***

* *HyperText Transfer Protocol:* communication protocol used to transfer data between the server and the client.
* *JSON:* JSON strings are used to transfer the data between the Java Server and JavaScript Client.

# Database

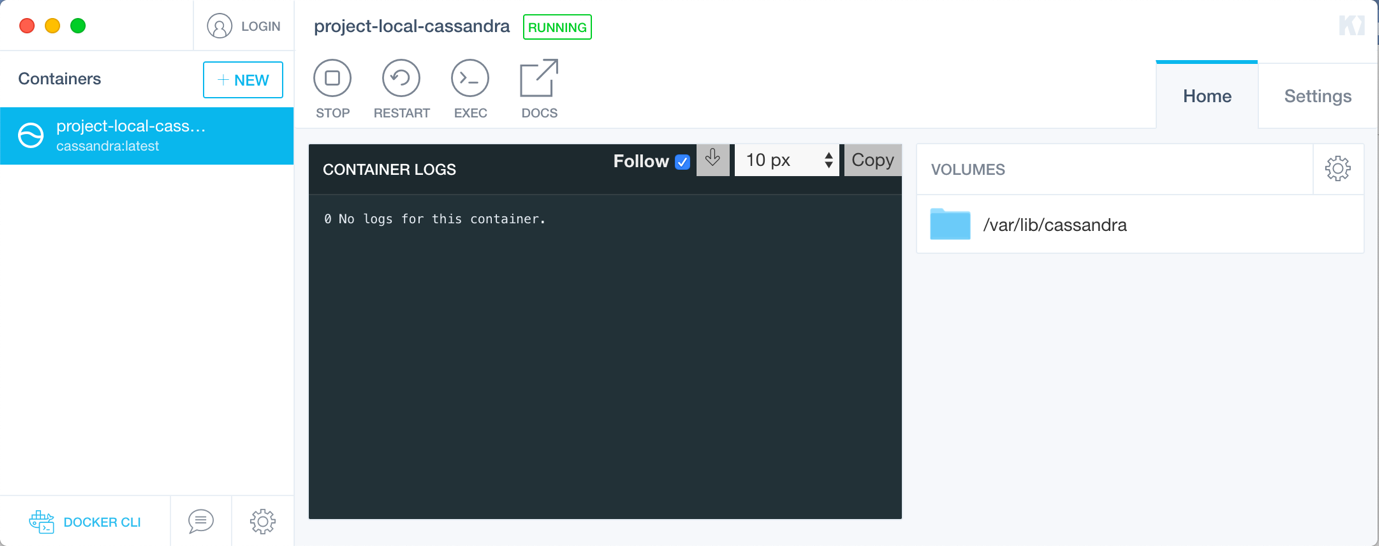
## Apache Cassandra

Apache Cassandra [1] is an open-source NoSQL distributed database system that is scalable, fault-tolerant and provides a high level of performance. The system is linearly scalable and is designed to handle large volumes of DB writes. These qualities make it suited to use with an application that anticipates a rapid expansion of the number of users of the application/system that submit data on a large scale. Cassandra operates with its data stored across nodes. The data is replicated across these nodes within what is known as a ‘cluster’ – which is a collection of nodes. This means the DB is not centralised and as such there are no single points of failure. Outage in one datacentre therefore does not mean data loss or system failure. Another node located at another datacentre takes over and provides what is needed.

## Docker

Withn this project I have integrated the software, Docker [2]. Docker is a software development tool used to create containers within which an application can be deployed and run. Its strength of use is that it allows a developer to deploy an application – along with its required libraries and dependencies within an isolated container environment. Docker does not require an entire operating system to be created using a virtual machine environment, but instead only draws the needed system resources from the native OS that Docker is running on. This project deploys a database instance of Cassandra within a container using Docker.

Kitematic [3] is a GUI application for viewing and organising Docker containers. Figure 6-1 is a screenshot of Kitematic, displaying the container being used for this project:



*Figure 6-1*

## Installation and Configuration

For much of the setup for the database and associated tools, the Command Line Interface (CLI) was used. This was carried out on a MacBook which uses a native package installer called Homebrew. With Homebrew, installation of any desired packages/tools can be done in just one or two commands.

Using Homebrew, Docker and Apache Cassandra were installed. Python was installed as well (needed for running the Cassandra Query Language Shell). With these elements installed, the next step is to setup the instance of the DB.

An important aspect of setting up Cassandra within Docker is port publishing - to allow for contact to and from external sources. This cannot be modified after container creation. This port configuration must be done at the time of creating the container with the Cassandra image from the Docker Image Library. Setup and correct configuration of the Cassandra container can be accomplished in a single command:

* docker run -p 9042:9042 --name cassandra -d project-local-cassandra

This command will create a new container called “project-local-cassandra’ with the ‘cassandra’ Docker image from the Docker library and publishes the port 9042 for communication.

## Keyspace and Table Setup

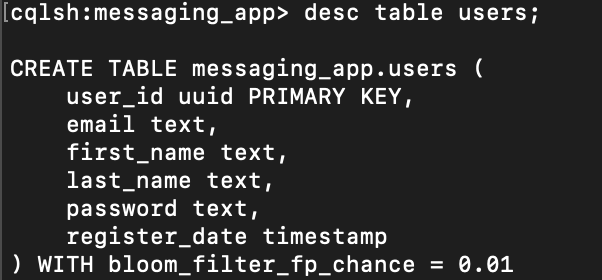
The language used to interact with Cassandra is the Cassandra Query Language (CQL). CQLSH is a command line shell that allows the user to manually access the locally hosted Cassandra DB instance. Using the CLI the user can input the command ‘cqlsh’ and an interactive shell (powered by a Python script) is launched. This shell can receive commands in CQL to read or write to tables found in the DB instance.

For this project a keyspace is required in which to store the necessary tables for the application’s data. This is done with the CQL command:

* CREATE KEYSPACE messaging\_app;

Following this, the tables must be setup. For this application three tables were decided upon.

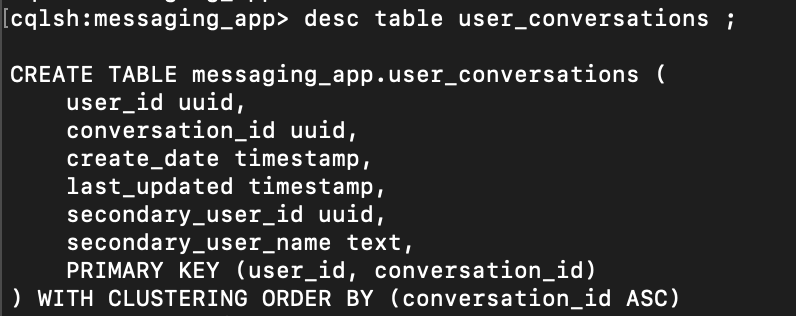
1. Table name: users



*Figure 6-2*

This table stores the details of a user: the name, email, password and date that the user registered. The Primary Key of the table is a Universally Unique Identifier (128-bit identifier) generated upon entry of the user to the DB.

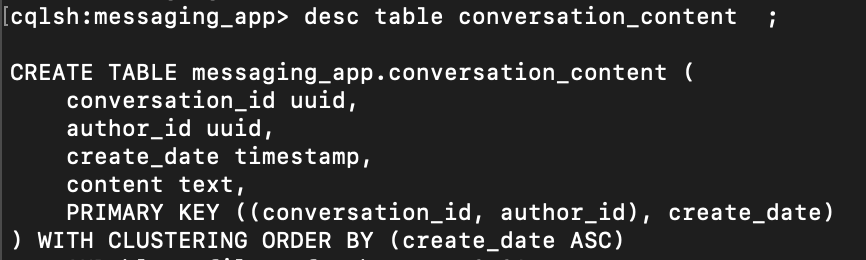
1. Table name: user\_conversations



*Figure 6-3*

Stores the conversations of the application. Using the compound Primary Key of user\_id and secondary\_id the server can determine the conversation\_id to be retrieved. This conversation\_id is then used in the third DB table to retrieve the content (messages) of that particular conversation between two users.

1. Table name: conversation\_content



*Figure 6-4*

Stores the messages/content of a particular conversation.

# Server

## Apache Tomcat

The responsibilities of this application’s server are to receive and process HTTP requests sent by the application’s client. In addition, the server must be capable of accessing and modifying the data stored in the Cassandra database. The requests sent from the client can include: inserting a message into a conversation, authenticating a user, registering a user, creating a conversation with another user.

The technology chosen to fulfil this purpose is Apache Tomcat [4]. Tomcat is a well-known and widely-used Java application server used to implement servlets. It is open-source software. This project implements the Java EE (Enterprise Edition) of the Tomcat application server. The software uses webservlets to service HTTP requests made to the server. Tomcat can be deployed with IntelliJ and this allows the efficient debugging of the code when designing and testing. It can also be deployed using the project Tomcat WAR file through the command line.

## Tomcat – Cassandra Connection

A critical part of the application is the connection between the server and database. With Tomcat the java code to establish this connection is relatively simple and short as seen in figure 7-1.



*Figure 7-1*

The difficulty with this task was in the complex configuration setup that is required for a local instance of Cassandra to communicate with an external source. There are a number of libraries and JAR files that are required. To streamline and make the process straightforward Apache Maven [5] was used. The Java project automation tool added the necessary dependencies needed for simple configuration of Cassandra.

## WebServlets

In addition to communicating with the database, the server handles incoming client requests. This functionality is achieved by the use of Tomcat’s web servlets. A single servlet will receive an HTTP request based on the URL pattern specified in the servlet path. For example, the Tomcat server running on the localhost machine may have a URL address of ***http://localhost:8080/tomcat\_server\_war\_exploded/*** and a particular servlet for this application has a URL pattern of “/RegisterUser”. Upon receiving the HTTP request addressed to “***http://localhost:8080/tomcat\_server\_war\_exploded/RegisterUser***/” to the server, Tomcat will send the request to whichever servlet matches the pattern (in this case the RegisterUser servlet) and process it. This is done with the @WebServlet java annotation as seen in figure 7-2.



*Figure 7-2*

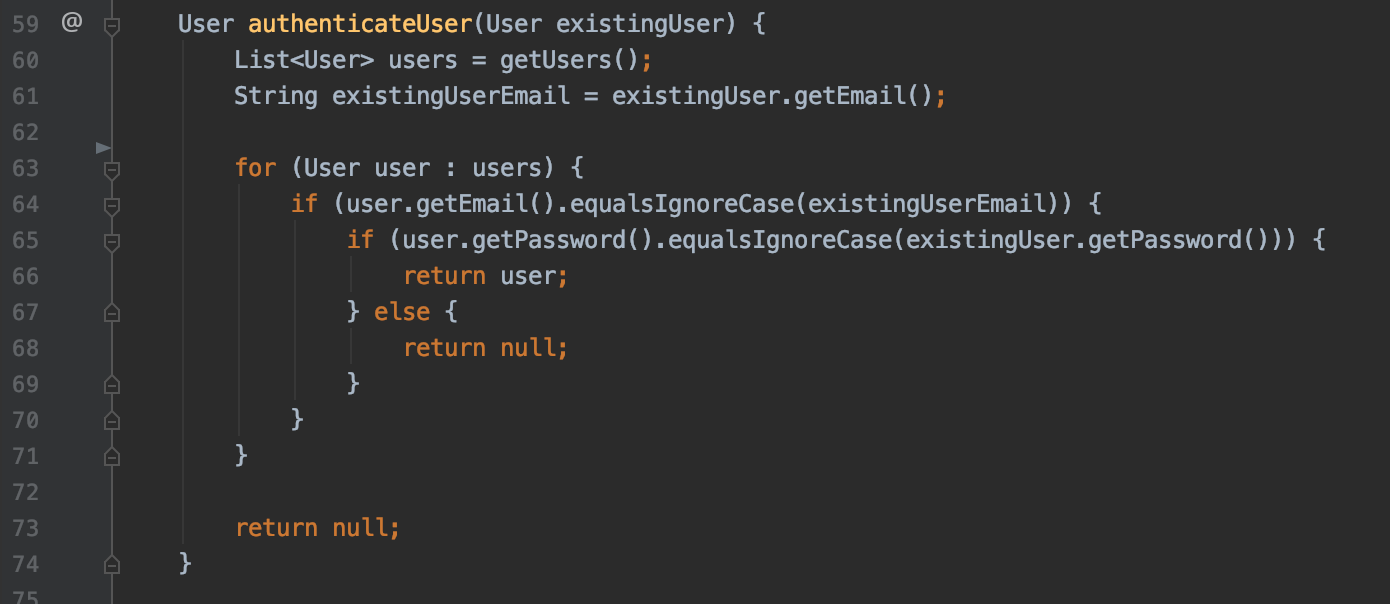
For this application a servlet has been designed to deal with each of the possible requests that can be made from the client.

* RegisterUser
  + Used to register a new user to the system
  + Provides basic input validation of email and password input
  + Checks if user is already registered
  + Servlet reads/writes to users DB table



*Figure 7-3*

* AuthenticateUser
  + Used to authenticate the login details provided by user
  + Basic input validation of input email and password
  + Checks if user is in system and password is correct
  + If verification process is successful servlet returns list of conversations the user is a part of
  + Reads/writes to users DB table



*Figure 7-4*

* GetUserConversations
  + Used to retrieve a list of conversations that a user is a part of
  + Reads from user\_conversations DB table



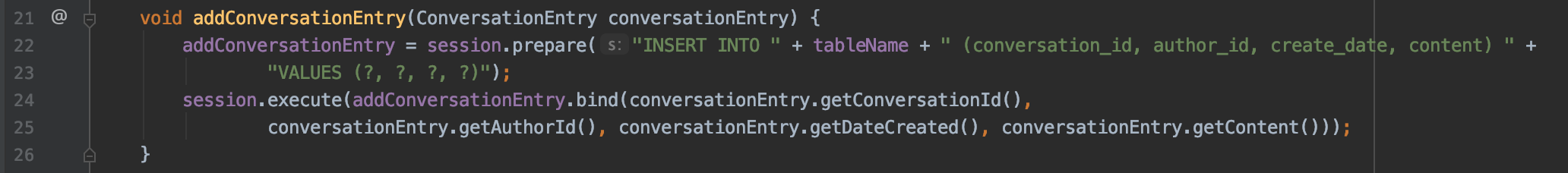
*Figure 7-5*

* GetConversationEntries
  + Used to retrieve a list of message entries for a particular conversation based on the provided user ID
  + Reads from conversation\_content DB table



*Figure 7-6*

* AddConversationEntry
  + Used to add a message entry for a conversation
  + Writes to the conversation\_content DB table



*Figure 7-8*

# Client

## React

For the design of the client element of this project, React [6] was chosen. React is a JavaScript library that is used for building and rendering user interfaces. It is a widely popular front-end framework for developers. The software is open-source and is maintained by Facebook. It is geared towards single-page applications.

## JSX and Virtual DOM

React uses a special language named JSX [7] to program the applications and features. It does not use regular JavaScript but a special type of JavaScript where HTML tag syntax is used to render subcomponents. This syntax is processed into JavaScript by the React runtime engine.



*Figure 8-1*

*An example of JSX syntax from the LoginForm component*

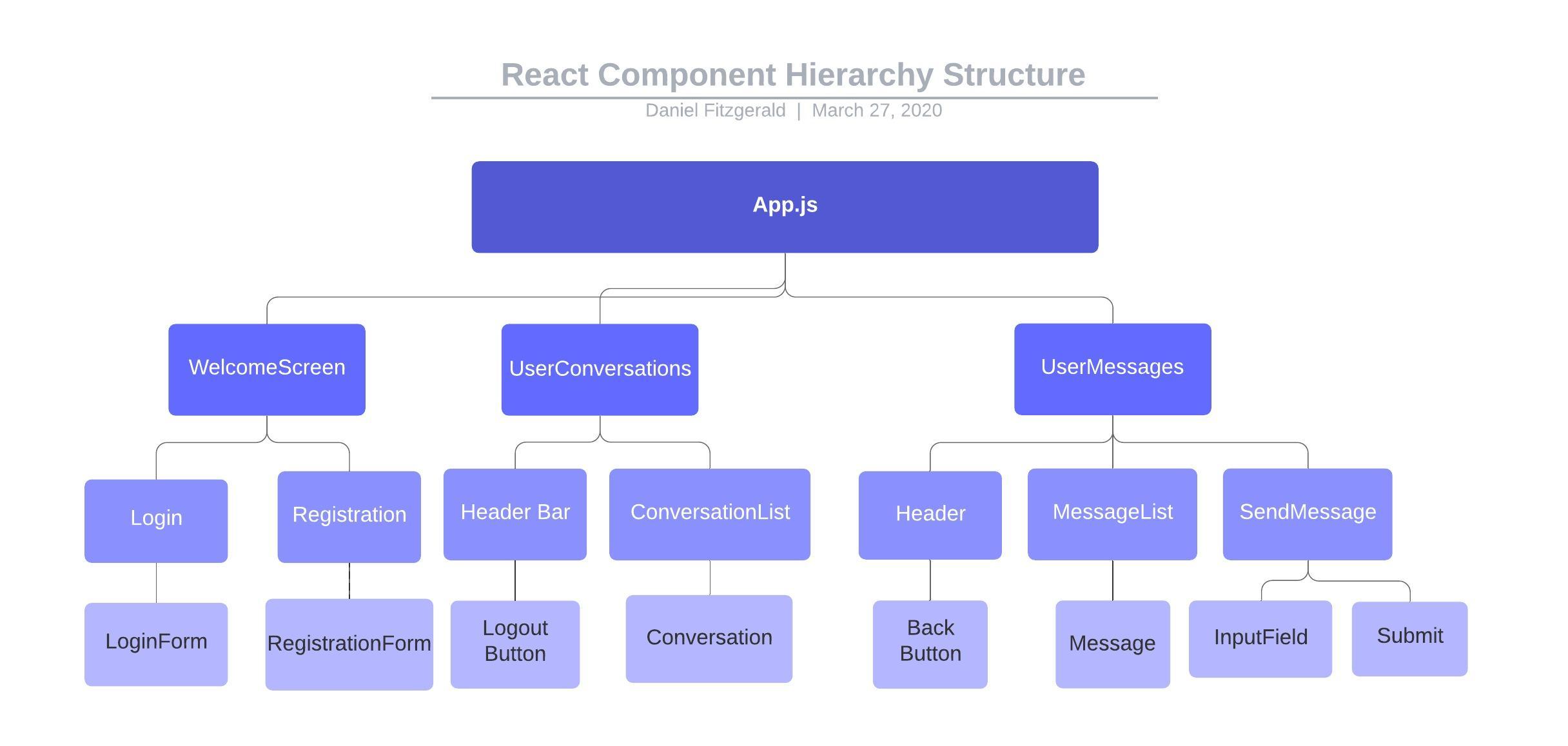
Another important feature of React is the Virtual Document Object Model (Virtual DOM). The DOM represents the UI of the application. Any time there is a change to the state of the UI, the DOM is updated and re-renders to represent the change. Many updates would mean a negative effect on performance. The virtual DOM is a virtual version of the real DOM. The Virtual DOM works by calculating the best possible method of making changes to the real DOM when a state change has occurred in the application. The logic involved ensures that minimal re-renders take place, thus improving performance.

## Components

One of the core and most important aspects of React is the creation of custom-made **components.** These components are like HTML elements that can be reused in a number of different ways to create a user interface. The components have special attributes that can store component state and properties. These attributes and data can be passed to other components as needed.

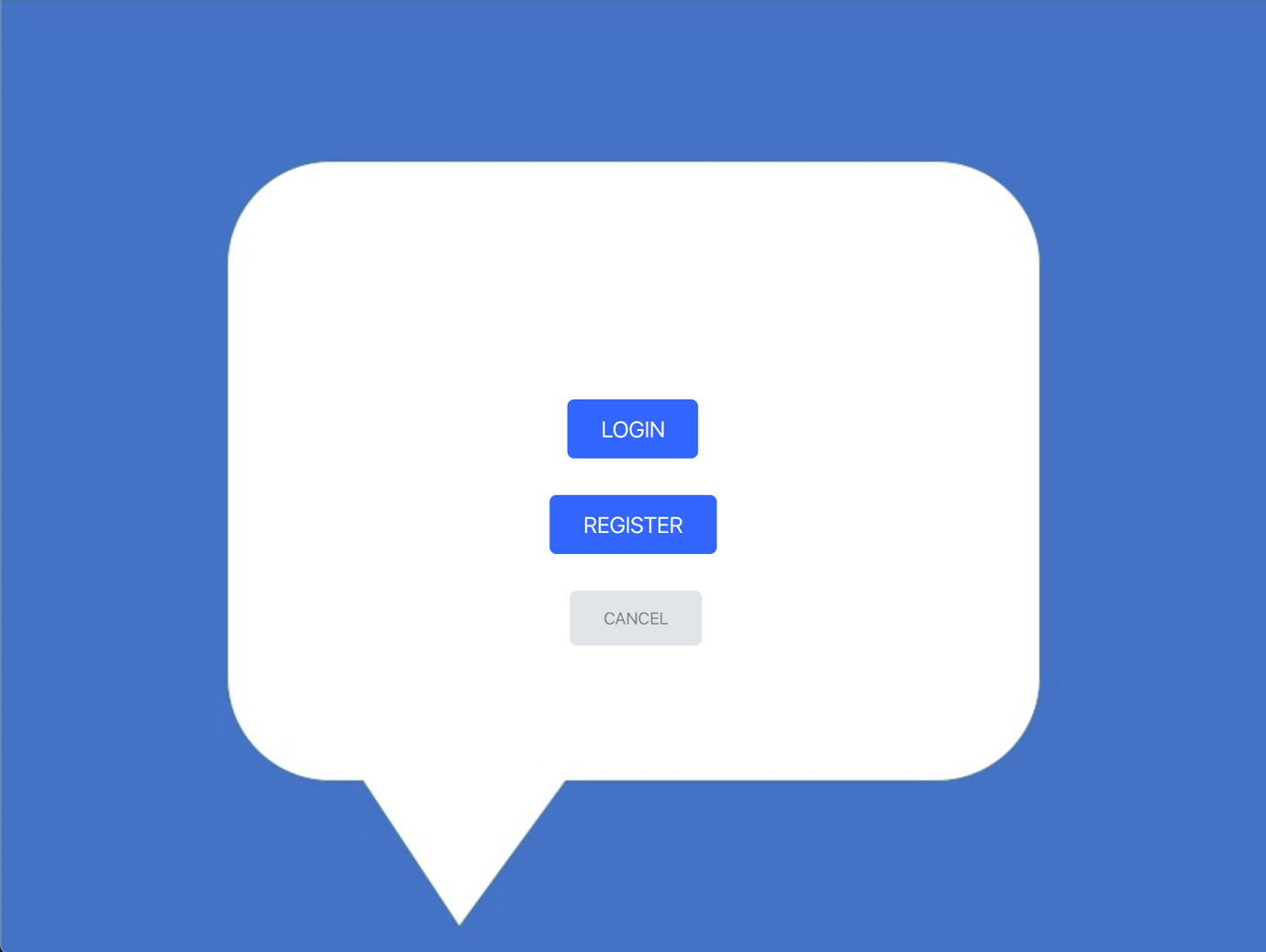
Components can be either functional or class components. If storing the state of a component is needed then a class component is used, otherwise a functional component is used. Class components have a parent-child hierarchy architecture.

For this application the client user interface was created with three primary components, each of which has multiple child components responsible for providing the needed elements to create the UI.



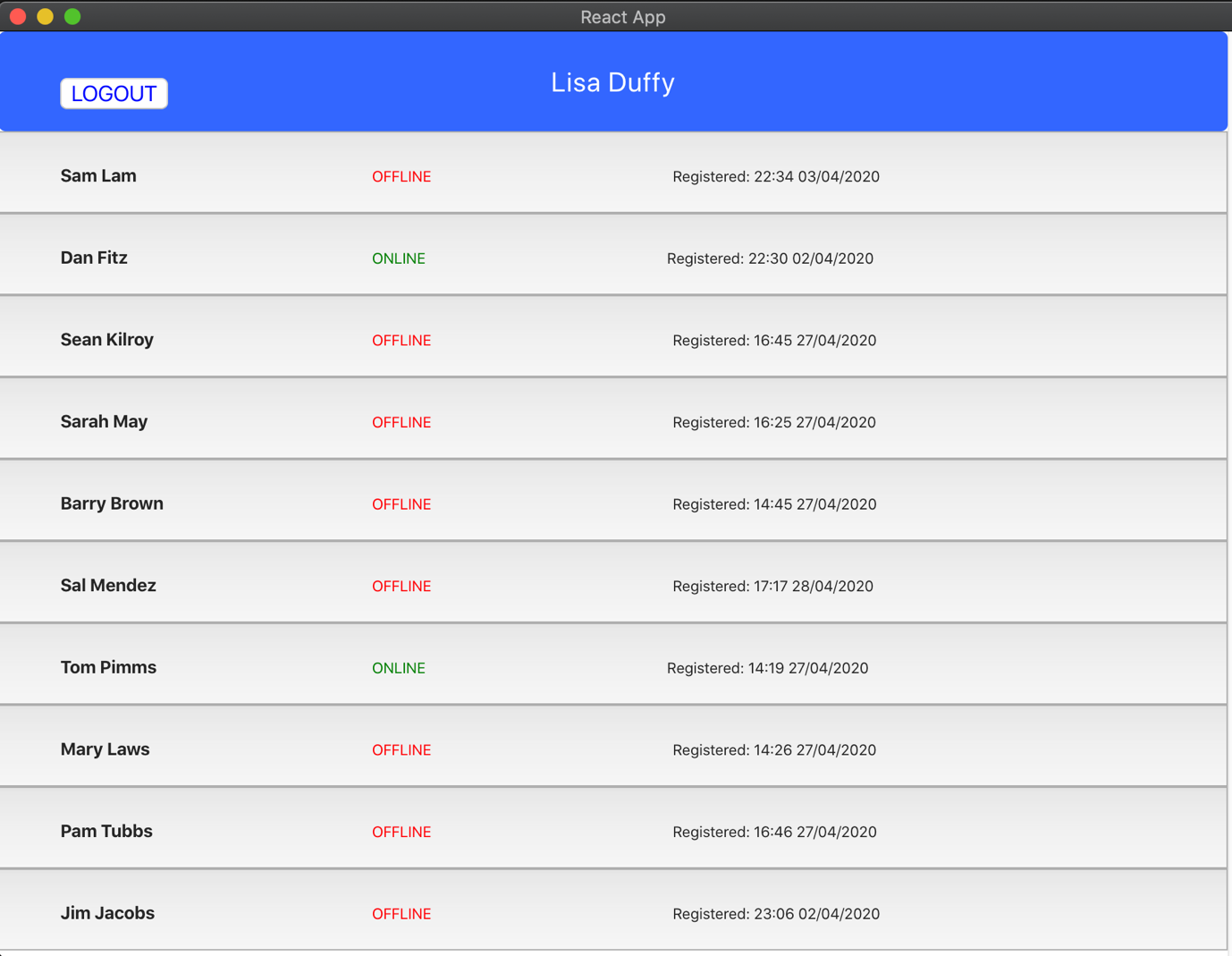
*Figure 8-2*

The first primary component to be rendered is the WelcomeScreen component. This is the component rendered on start-up of the application. It presents the user with the Login and Register components. These will render the necessary user input form depending on the user’s choice. All components contain the necessary JavaScript code to send the relevant HTTP request to the server based on user actions. Figure 8-3 shows the welcome screen.



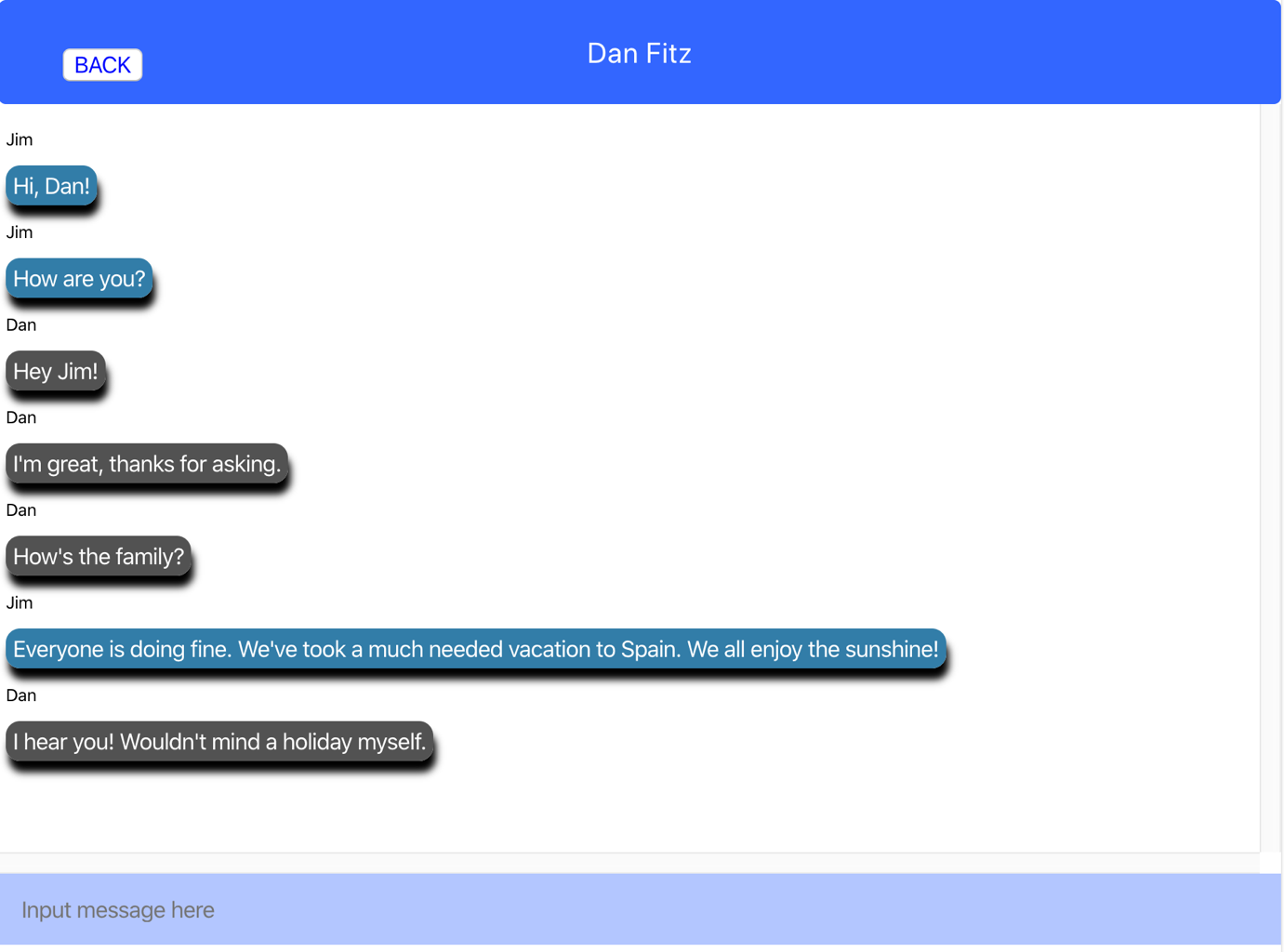
*Figure 8-3*

Following the successful login/registration of the user the next component rendered is UserConversations. The user is presented with a list possible conversations. This information is the result of retrieving all users in stored in the users table in the database. Figure 8-4 shows the conversation list screen.



*Figure 8-4*

The final component is the MessageList component. When a conversation is chosen by the user, all messages belonging to that particular conversation are displayed on the UI.



*Figure 8-5*

All components contain the necessary code to send the relevant HTTP request to the server based on user actions.

# Network

Networking is an essential part of this application. The two aspects of the application’s networking capability are between server and database and between server and client. The networking functionality between server and database is already covered in section 7.2.

## Server and Client

In order for the application to fulfil its operational requirements, the client and server must be capable of sending and receiving data between each other. The communication protocol used in this project is HyperText Transfer Protocol. The React components send relevant HTTP requests to the server when certain events occur (login request, sending a message etc.). The server implements web servlets to receive and process these requests.

In the client code, an AJAX (Asynchronous JavaScript and XML) [8] object called XMLHttpRequest is used to send a request.



*Figure 9-1*

*Code snippet of XMLHttpRequest being used to register a user*

The request is sent to the relevant Tomcat web servlet by using the URL address in request.open().

When the server receives the request it processes the incoming data and sends back a response to the client.

## JSON

As the server of this application is written in and implemented by Java and the client is written in React JavaScript, the two must communicate using a single uniform medium. Java objects cannot be interpreted by JavaScript and vice-versa. JSON (JavaScript Object Notation) [9] is the solution to this problem. When writing and outputting a response to the client, the server will take the outgoing data and convert it to a JSON string. This is then returned, as seen below in figure 9-2.



*Figure 9-2*

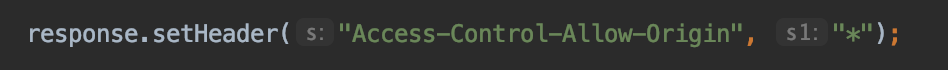
The client receives a response from the server containing the retuned data within the request body. The incoming JSON string from the server is parsed to resolve the string into a JavaScript object. The data can now be used in the client application.

# System Integration

Multiple stages throughout the project’s development required integration of new software and technologies. The first stage involved integrating the necessary applications and tools to establish the database instance and enable communication through the command line.

The next significant step was to successfully connect my java server application with the DB. This proved very difficult when attempts were made to add the necessary JARs and dependencies to the project manually. Apache Cassandra and Tomcat EE are enterprise-level applications and require complex configuration. This problem was overcome by creating a Maven project and migrating my server code to the new project. Maven’s automated project management features allowed simple and straight-forward adding of dependencies to the project structure. This resulted in a configured Cassandra-Tomcat integration.

The integrating of server and client presented a difficult problem to solve. As React runs on localhost port:3000 and Tomcat runs on localhost port:8080, the React client (which runs from the browser) implements a policy called Cross Origin Resource Sharing (CORS) [10]. This policy is a web security measure designed to prevent cross-domain hacking. The issue required much research to find a solution. A particular line of code was added to the web servlets in order to allow a browser from a different origin to access the server’s resources. The “Access-Control-Allow-Origin” header within the response has its value set to “\*”. This allows the client the needed access.



*Figure 10-1*

# Conclusion

The result of this project is a functioning full-stack messaging application that is capable of client - server - database communication. The application allows users to exchange text messages to each other in a chat environment.

Database reads and writes can be achieved by manual manipulation by the command line interface or by java code executed within Tomcat web servlets. The project’s server functionality can handle a number of different requests from a client and return whatever values needed from the DB. The React client successfully renders a user interface that can be interacted with by a user. It can send HTTP requests to the server and process the response data, rendering the necessary components to the UI and displaying the results and data.

The goal of this project was to plan, design and build a full-stack messaging chat application. This has been successfully accomplished. In addition, throughout the process of this endeavour many advancements have been made in personal skills and technical knowledge over a range of software development areas. These include learning how to write and design original source code for both java web servlets and React components. There has been a substantial increase in database knowledge, not just in query language skills but in understanding database layout and tabling formats. The difficulties encountered – especially with integration - produced an improvement in problem solving skills along with a deeper understanding of networking.

# References

[1] Apache Software Foundation, “Apache Cassandra”, <http://cassandra.apache.org>, 2020

[2] Docker, <https://www.docker.com>, 2020

[3] Kitematic by Docker, <https://kitematic.com>, 2020

[4] Apache Software Foundation, ”Apache Tomcat”, <http://tomcat.apache.org>, 2020

[5] Apache Software Foundation,”Apache Maven”, <http://maven.apache.org>, 2020

[6] React, <https://reactjs.org>, 2020

[7] “Introducing JSX”, <https://reactjs.org/docs/introducing-jsx.html>, 2020

[8] “Ajax – The XMLHttpRequest Object”, <https://www.w3schools.com/js/js_ajax_http.asp>, 2020

[9] “JSON W3Schools”, https://www.w3schools.com/js/js\_json\_intro.asp , 2020

[10] “What is CORS?”, <https://www.codecademy.com/articles/what-is-cors>, 2020